ARBEITSBERICHTE
DES INSTITUTS FÜR MATHEMATISCHE MASCHINEN UND DATENVERARBEITUNG (INFORMATIK)
FRIEDRICH ALEXANDER UNIVERSITÄT ERLANGEN NÜRNBERG
HERAUSGEBER: H. Billing, W. Händler, U. Herzog, F. Hofmann, K. Leeb, P. Mertens, H. Müller, G. Nees, H. Niemann, D. Seitzer, B. Schmidt, H. J. Schneider, H. Wedekind

# FELDRECHNER DAP: INVERTIERUNG GROSSER MATRIZEN

Werner Erhard

DISSERTATION
BAND 19 · NUMMER 1 · ERLANGEN · FEBRUAR 1986

1986 № 72

ISSN 0344 - 3515

Digitized by the Internet Archive
in 2026

https://archive.org/details/IA41538124_0020

# FELDRECHNER DAP:
# INVERTIERUNG GROSSER MATRIZEN

Der Technischen Fakultät

der Universität Erlangen-Nürnberg

zur Erlangung des Grades

DOKTOR - INGENIEUR

vorgelegt

von

Werner Erhard

ERLANGEN 1986

Als Dissertation genehmigt von der
Technischen Fakultät der
Universität Erlangen-Nürnberg

Tag der Einreichung: 21.10.1985
Tag der Promotion: 13.1.1986
Dekan: Prof. Dr. Ing. H. Brand
Berichterstatter: Prof. Dr. Ing. U. Herzog
Prof. Dr. F. Hofmann

Mein besonderer Dank gilt:

Herrn Professor Dr. U. Herzog für die Betreuung und Unterstützung dieser Arbeit

Herrn Professor Dr. F. Hofmann für die Übernahme des Zweitgutachtens

Herrn Dr. K.D. Reinartz für die Durchsicht der Arbeit und die vielen Anregungen

Allen Kolleginnen und Kollegen, die durch Kritik und Anregungen zum Gelingen dieser Arbeit beigetragen haben

KURZFASSUNG

In dieser Arbeit wird am Beispiel der Matrixinversion dargestellt, wie am DAP große Matrizen ( groß im Sinne von nicht komplett in den Speicher des DAP passend ) mit geringem Aufwand für den Benutzer bearbeitet werden können.
Dazu werden spezielle Eigenschaften des DAP untersucht und es wird geprüft, in wie weit sich vorhandene Algorithmen zur Lösung linearer Gleichungssysteme bzw. zur Matrixinvertierung effizient auf dem DAP implementieren lassen.
Zur Partitionierung von großen Matrizen wird in Abhängigkeit vom ausgewählten Invertierungsalgorithmus eine Abbildungsfunktion definiert, die eine optimale Zuordnung von Zeilen und Spalten der Matrix auf den DAP-Speicher vornimmt. Dabei wird auf eine schnelle Adressierungsmöglichkeit dieser Zeilen und Spalten Wert gelegt.
Als Ergebnis wird dem Benutzer ein Schema vorgegeben, mit dem er den Rechenzeitaufwand für beliebig große Matrizen abschätzen kann und mit dem er ohne zusätzlichen Programmieraufwand seine Matrizenbearbeitung vornehmen kann.

Inhaltsverzeichnis


1. Einleitung 5

2. Der Distributed Array Processor (DAP) 11

2.1 Überblick 11

2.2 Entwicklung des DAP 17

2.3 Die Architektur des DAP 18

2.4 DAP-Fortran 27

2.5 Kommunikation Host - DAP 29

2.6 Arithmetik auf dem DAP 32

3. Asynchrone Ein-/Ausgabe am DAP 34

3.1 Ablaufmodell 1 36

3.2 Ablaufmodell 2 38

3.3 Vergleich der Modelle 41

3.4 Messungen zur asynchronen Ein-/Ausgabe 44

4. Das Invertierungsverfahren von Sherman-Morrison und seine Modifizierung für den DAP 49

4.1 Invertierungsverfahren 49

4.2 Große Matrizen 52

4.3 Das Verfahren von Sherman-Morrison 53

4.4 Modifikation des Lösungsverfahrens 57

5. Die Abbildungsfunktion CRACKING 65

5.1 Crinkling 65

5.2 Slicing oder Partitioning 65

5.3 Cracking 68

5.4 3. Modifikation des Invertierungsverfahrens 71

6. Die Komplexität des Verfahrens 73

7. Ergebnisse 81

7.1 Testmatrizen 81

7.2 Implementierung 82

7.3 Meßergebnisse am DAP ohne asynchrone Ein-/Ausgabe 85

7.4 Vergleich mit Ergebnissen an anderen Rechnern 87

7.5 Fehlerabschätzung 90

7.6 Vergleich der theoretischen und praktischen Ergebnisse 92

7.7 Abschätzung der benötigten Rechenzeit 94

8. Asynchrone Ein-/Ausgabe und Sherman-Morrison-Verfahren 95

9. Zusammenfassung und Ausblick 100

Literaturverzeichnis 101

# 1. EINLEITUNG

In den letzten Jahren sind durch ständige Verbesserungen die konventionellen Rechensysteme immer leistungsfähiger geworden. Trotz dieser Weiterentwicklung gibt es bei der Bearbeitung großer Matrizen immer noch die beiden grundsätzlichen Probleme

- hoher Rechenzeitbedarf
- zu kleiner Arbeitsspeicher und damit hoher Kanalzeitbedarf

Die Lösung von großen Gleichungssystemen erfordert daher nicht selten CPU-Zeiten im Tagebereich und Verweilzeiten im Wochenbereich. Da man sich andererseits in den Ingenieurwissenschaften zunehmend an Aufgaben wagt, die die Lösung großer Gleichungssysteme erfordert, versucht man, auf neuartige Rechner auszuweichen. Dies ist insbesondere immer dann erforderlich, wenn die Matrizen dicht besetzt sind. Solche Matrizen treten z.B. bei strömungsmechanischen Berechnungen im Bereich von Turbulenzen auf.
Es hat sich gezeigt, daß sich die Struktur des DAP ausgezeichnet für die Lösung matrixähnlicher Probleme eignet.
Die Leistungsfähigkeit des DAP ist so hoch, daß die benötigte CPU-Zeit im Vergleich zu konventionellen Großrechnern ( z.B. Cyber 173 ) um eine Größenordnung sinkt.
Das bedeutet, daß am DAP nur noch CPU-Zeiten im Minutenbereich benötigt werden, wo an der Cyber 173 noch CPU-Stunden notwendig waren.

Damit ergeben sich bei der Bearbeitung großer Matrizen hinsichtlich des CPU-Zeitbedarfs kaum noch Probleme.
Dies ist eine wesentliche Verbesserung der Rechensituation bei zeitintensiven Berechnungen.
Da der DAP insgesamt jedoch nur über einen Arbeitsspeicher von 8 Megabyte verfügt, bleibt das Problem des meist zu kleinen Speichers.
Ziel der vorliegenden Arbeit war es, ein Berechnungsverfahren zur Lösung von großen Gleichungssystemen für den DAP zu entwerfen. Dabei wurde besonders Wert darauf gelegt, spezielle Architektureigenschaften des DAP in den Entwurf einzubringen.

Insbesondere war es erforderlich

- einen Invertierungsalgorithmus so zu entwerfen oder zu modifizieren, daß die Matrixstruktur des DAP-Prozessorfelds gut ausgenutzt wird
- die Möglichkeit des "cycle-stealing" durch asynchrones Ein-/Ausladen von Daten zu nutzen
- Invertierungsverfahren und asynchrone Datenübertragung aufeinander abzustimmen
- es einem Benutzer zu ermöglichen, die benötigte CPU-Zeit zur Invertierung seiner Matrizen vorher abzuschätzen.

Da in der Literatur eine Vielzahl von Verfahren zur Matrixinvertierung vorhanden sind, wurde versucht, durch Modifikation und Anpassung an die spezielle Struktur des DAP ein bereits vorhandenes Verfahren zu implementieren. Ausgewählt wurde ein 1949 von J. Sherman und W.J. Morrison publiziertes Verfahren [SHM049]. In der Arbeit wird gezeigt, daß es sich mit einigen Modifikationen und der Definition einer neuen Abbildungsfunktion von Daten auf den DAP-Speicher gut realisieren läßt.
Durch den knappen DAP-FORTRAN-Code ist der Programmieraufwand sehr gering. Die wesentliche Arbeit steckt in der logischen Synthese des Verfahrens und der DAP-Struktur.
Der Benutzer kann das komplette Lösungssystem in einfacher Art und Weise anwenden. Er kann ebenso die ungefähre Rechenzeit für die von ihm angegebene Matrixgröße anhand einer einfachen Formel abschätzen.
Wie bei allen anderen Verfahren auch, ist zu beachten, daß es durch schlecht konditionierte Matrizen eventuell zu groben Berechnungsfehlern kommen kann. Es kann deshalb auch hier nicht darauf verzichtet werden, klassische Fehlerabschätzungen vorzunehmen [WILK69].

In Kapitel 2 wird zunächst der DAP vorgestellt sowie seine Besonderheiten der Hard- und Software.
Die einzelnen Komponenten des DAP

- Prozessormatrix
- Speicher
- Master Control Unit

werden in ihrem Aufbau dargestellt.
Die für die Verarbeitung von Daten sich ergebenden Speichermöglichkeiten

- horizontal (Vektormodus)
- vertikal (Matrixmodus)

werden skizziert und ihre Vor- und Nachteile erläutert.
Für die Programmierung des DAP wurde Fortran 4 erweitert um die Datentypen Vektor und Matrix sowie um zusätzliche Funktionen zur effizienten Bearbeitung dieser neuen Datentypen.
Die arithmetischen Operationen und die Standardfunktionen wurden auf diese Datentypen erweitert. Abschließend wird in diesem Kapitel auf die schnelle Berechnung der Standardfunktionen am DAP hingewiesen.
Die Möglichkeit, asynchron zur Bearbeitung eines Programms im DAP Daten in den Speicher nachzuladen bzw. auszulagern, wird in Kapitel 3 dargestellt. Man macht sich hier die unterschiedlichen Zykluszeiten von Prozessor und Speicher zunutze. Dieses Verfahren ist unter dem Begriff "cycle-stealing" bekannt.
Anhand eines Modells wird die Wirkungsweise dieser asynchronen Ein-/Ausgabe erläutert und auf ihre Schwachstellen eingegangen. In einem zweiten Modell wird anschließend ein Vorschlag zur Beseitigung dieser Probleme gemacht.

Das Verhalten der asynchronen Ein-/Ausgabe und eventuelle Einflüsse auf laufenden Programme im DAP werden durch Meßreihen analysiert.
Kapitel 4 befaßt sich mit einem speziellen Verfahren zur Lösung linearer Gleichungssysteme bzw. zur Invertierung von Matrizen. Dabei wird in jedem Iterationsschritt genau eine Spalte oder Zeile der zu invertierenden Matrix benötigt. Dieses Verfahren wird auf die Struktur des DAP abgebildet und durch Modifikationen optimiert, die zum Ziel haben,

- die Komplexität des Verfahrens zu verkleinern und
- die Datenmenge überschaubar und einfach adressierbar zu machen.

Da in der Literatur die Beweise für die Richtigkeit des Invertierungsalgorithmus fehlen, werden sie an dieser Stelle geführt.
Zur Implementierung war es notwendig eine Abbildungsfunktion zu definieren. In der Literatur sind das Crinkling und das Slicing bekannt. Als Weiterentwicklung des Slicing wird das als Cracking bezeichnete Verfahren in Kapitel 5 beschrieben.
Um das Verfahren bewerten zu können, muß man zunächst theoretische Aussagen über die Komplexität machen. In Kapitel 6 wird anhand der Anzahl der notwendigen Multiplikationen und Additionen ein Komplexitätsmaß für das Verfahren mit seinen Modifikationen angegeben.

Eine Bewertung des Verfahrens wird in Kapitel 7 vorgenommen. Die in Kapitel 6 erzielten theoretischen Ergebnisse werden durch die praktischen Messungen bestätigt. Die Meßergebnisse am DAP werden auch mit Ergebnissen an der Cyber und an der EGPA-Pyramide verglichen. Es wird eine Formel abgeleitet, nach der der Benutzer seinen Rechenzeitbedarf am DAP bei der Bearbeitung seiner Matrizen abschätzen kann.
Schließlich wird in Kapitel 8 eine Synthese zwischen der asynchronen Ein-/Ausgabe und dem Invertierungsverfahren vorgenommen und bewertet. Es wird angegeben, wieviel Zeit gegenüber dem synchronen Fall eingespart werden kann.
Abschließend erfolgt ein Ausblick, wie das vorhandene System zur Matrixinvertierung weiter ausgebaut werden könnte,
um zu einem umfassenden Werkzeug für die Matrizenbearbeitung (Multiplikation, Eigenwertaufgaben usw.) zu kommen.

# 2. Der Distributed Array Processor (DAP)

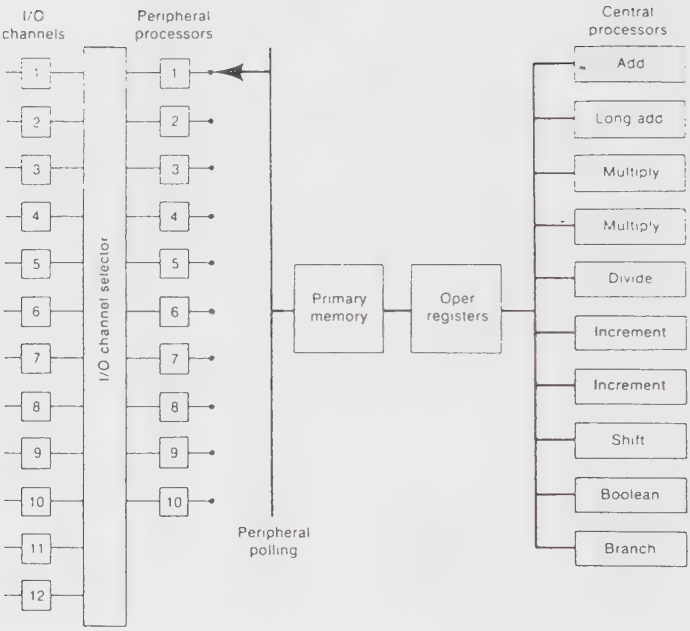
## 2.1 Überblick

Um von der klassischen von-Neumann-Architektur zu Rechnern mit Parallelverarbeitung zu kommen, ist man in der Vergangenheit verschiedene Wege gegangen. Alle hatten jedoch zum Ziel, statt einer mehrere Operationen gleichzeitig ablaufen zu lassen.

Man unterscheidet im wesentlichen 4 verschiedene Prinzipien der Parallelverarbeitung:

### 2.1.1 Funktionale Trennung

Der Rechner hat nicht ein einziges Rechenwerk, sondern für verschiedene Tätigkeiten verschiedene Funktionseinheiten (z.B. für logische Operationen,für Addition, für Multiplikation). Diese Funktionseinheiten arbeiten gleichzeitig auf unterschiedlichen Daten (z.B. CDC 6600).
Das folgende Bild von der Struktur der CDC 6600 verdeutlicht dies:

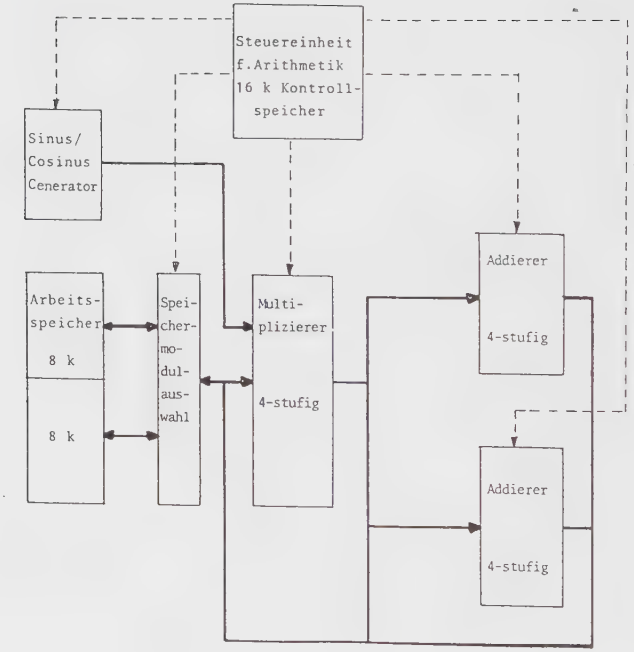



CDC 6600 mit verschiedenen Prozessoren

für verschiedene Aufgaben

### 2.1.2 Pipeline

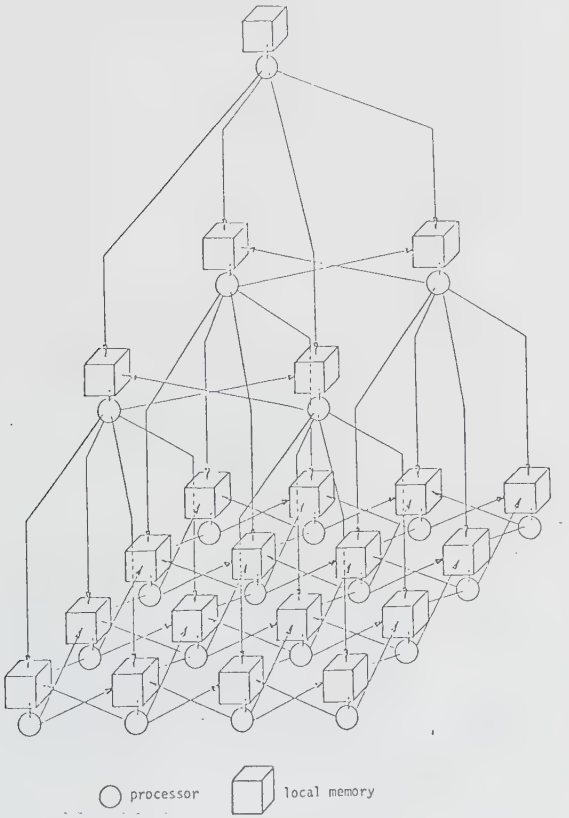
Die einzelnen Operationen wie z.B. die Gleitpunktaddition werden in kleine Elementarschritte ( Trennung Mantisse-Exponent, Angleichen der Mantissen, Addition, Normierung ) zerlegt und hintereinander ausgeführt. Ist der Pipeline-Vorgang einmal eingeschwungen, so steht nach jeder Taktzeit ein Ergebnis zur Verfügung. Diese Rechner eignen sich für die Ausführung gleicher Operationen auf großen Datenmengen.
Oftmals wird das Pipeline-Konzept ergänzt durch das funktionale Trennen wie bei 2.1.1. Dadurch erhält man mehrere Pipelines, die unabhängig voneinander arbeiten können (z.B. Cyber 205,Cray 1, IBM 3838).




Struktur der IBM 3838

### 2.1.3 Multiprozessoren

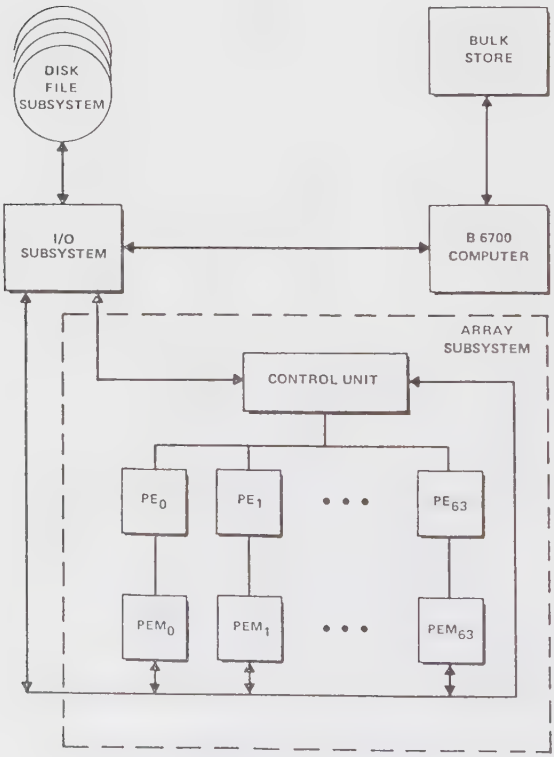
Mehrere Prozessoren, die nicht unbedingt vom selben Typ sein müssen, führen im allgemeinen unterschiedliche Programme aus. Sie kommunizieren untereinander beispielsweise über gemeinsame Speicher ( MIMD-Rechner, z.B. EGPA ).




Struktur der EGPA-Pyramide

### 2.1.4 Feldrechner

Eine große Anzahl identischer Prozessoren führt gleichzeitig auf unterschiedlichen Daten dieselbe Operation aus (SIMD-Rechner, z.B. Illiac IV,DAP,STARAN,BSP).




Struktur der ILLIAC IV

## 2.2 Entwicklung des DAP

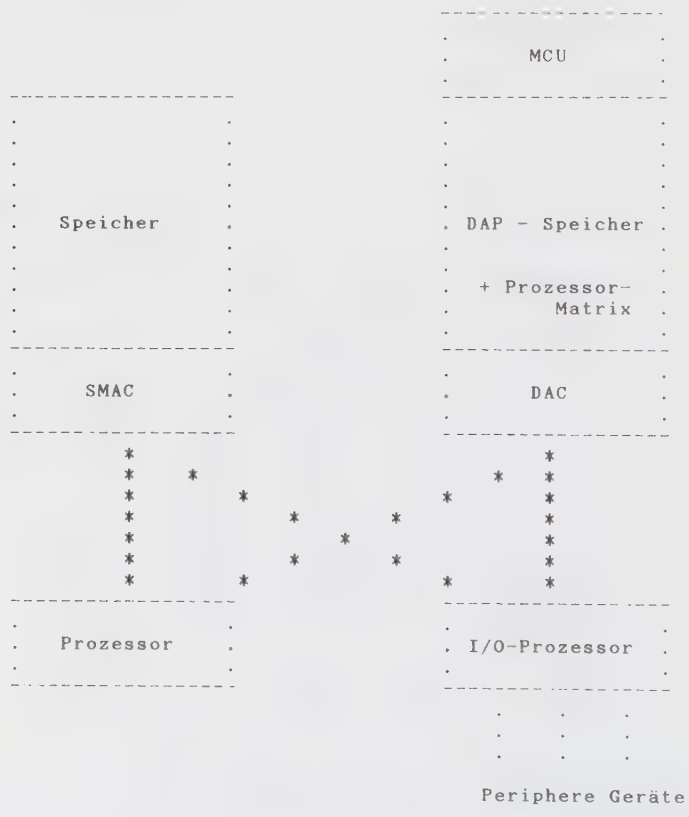
Im Jahre 1972 stellte man bei der Firma ICL in England erste Überlegungen an zum Entwurf eines Parallelrechners.
Das Ergebnis war ein aus einfachen und deshalb billigen TTL-Bauteilen aufgebauter Rechner, der 2 Besonderheiten aufweist:

1. Die Prozessoren sind zweidimensional in Matrixform angeordnet und können mit ihren Nachbarn kommunizieren.
2. Die Daten werden vertikal im Speicher abgelegt und bearbeitet. Damit ist eine Bitadressierung auf einfache Art und Weise möglich.

1976 wurde ein erster Prototyp mit 32 x 32 Prozessoren hergestellt, der nur in Maschinensprache programmiert werden konnte.
Die Programmierung in der Assemblersprache APAL war jedoch zu mühsam, so daß 1978 ein DAP-FORTRAN-Compiler fertiggestellt wurde.
1980 wurde dann der erste Rechner ausgeliefert. Das Queen-Mary-College in London erhielt einen DAP mit 64 x 64 Prozessoren und 4 kbit Speicher pro Prozessor. Dieser Speicher erwies sich später als Engpaß und wurde deshalb vergrößert. Jeder Prozessor hat jetzt 16 kbit Speicher, insgesamt verfügt der DAP damit über einen 8 MByte-Speicher.

## 2.3 Die Architektur des DAP

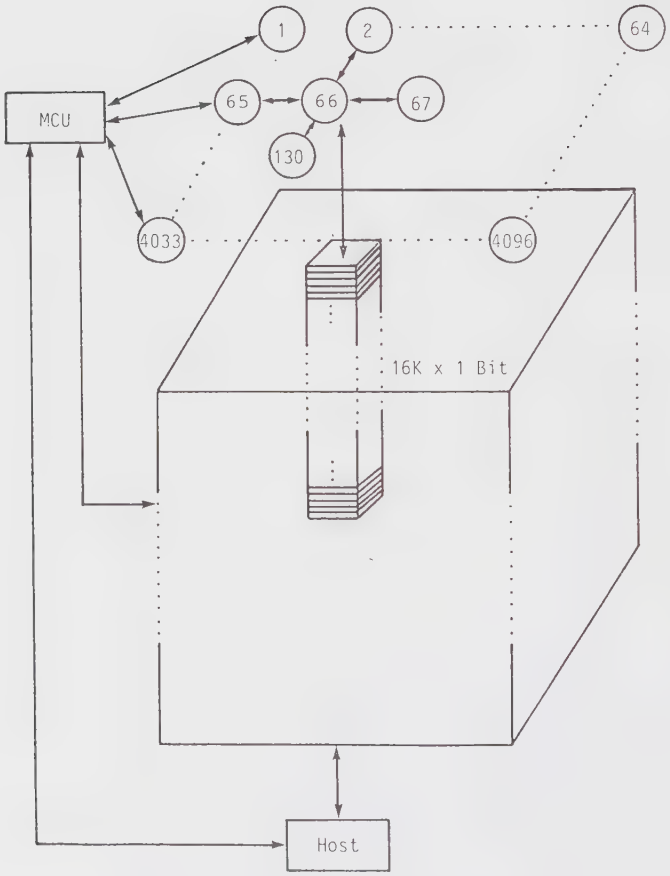
Der DAP ist kein stand-alone-Rechner. Er braucht als Host einen Rechner der 2900-Familie von ICL. Der Host ist notwendig für Datenhaltung, Ein-/Ausgabe und das Übersetzen, Binden und Starten der Programme.



Die Abbildung zeigt die typische Konfiguration eines 2900-Systems mit DAP.

Prozessor und I/O Prozessor können jeweils über Controller ( SMAC = Storage multiple access controller, DAC = DAP access controller ) sowohl auf den normalen Arbeitsspeicher als auch auf den DAP-Speicher zugreifen. Ist das Prozessorfeld des DAP abgeschaltet, dann kann der DAP-Speicher als Arbeitsspeicher für den Host verwendet werden.

Betrachtet man jetzt den DAP alleine, so ergibt sich folgende detaillierte Struktur:



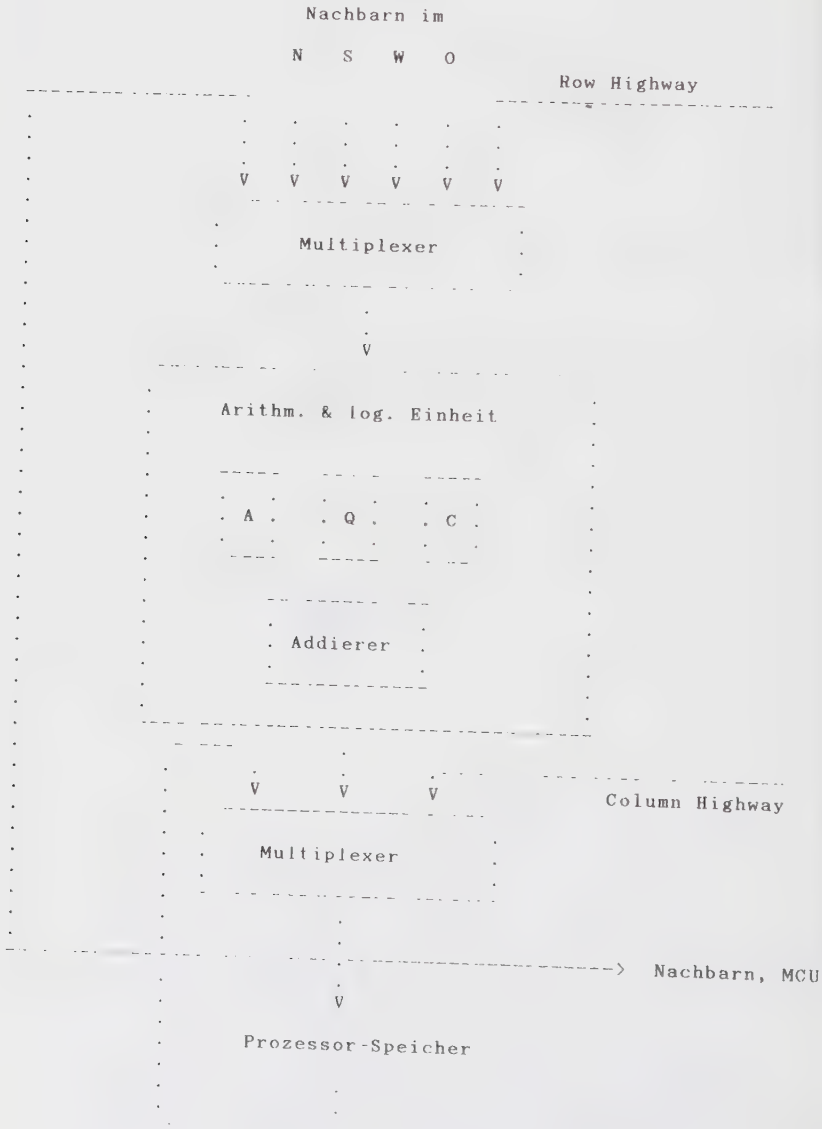
Der DAP besteht im wesentlichen aus 3 Teilen :

1. Prozessorfeld
2. Speicher
3. Master Control Unit (MCU)

### 2.3.1 Das Prozessorfeld

Das Prozessorfeld des DAP besteht aus 4096 Prozessoren, die in einer 64 x 64 - Matrix angeordnet sind. Jeder Prozessor hat Zugriff zu seinem eigenen Speicher ( 16 K x 1 Bit ) und zu seinen 4 Nachbarn im Norden, Süden, Westen und Osten.
An den Rändern der Prozessormatrix muß man sich die Verbindungen zu den Nachbarn fortgesetzt zu den gegenüberliegenden Rändern vorstellen. So hat z.B. Prozessor Nr. 64 als nördlichen Nachbarn Prozessor Nr. 4096 und als östlichen Prozessor Nr. 1 (siehe Bild S. 20)

Die Prozessoren selbst sind sehr einfach aufgebaut:



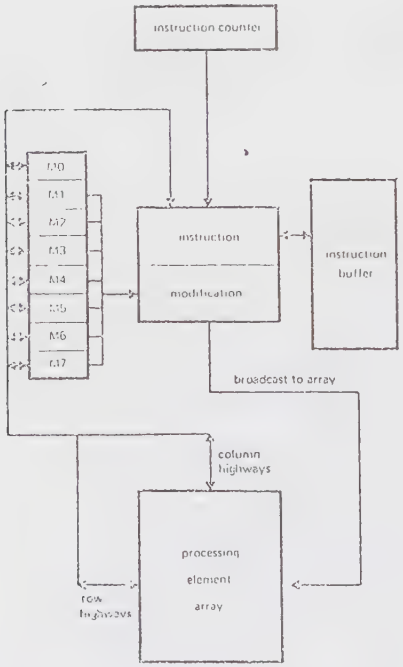
Es handelt sich um 1-Bit-Prozessoren, die über 3 Register A (Activity), Q (Accumulator) und C ( Carry) und einen Addierer verfügen. Der Addierer verknüpft die Register Q, C und den Eingang immer dann miteinander, wenn das Register A den Wert True hat. Damit ist es möglich, über Masken einzelne oder auch mehrere Prozessoren stillzulegen.
Als Eingang wird über einen vorgeschalteten Multiplexer das Ergebnis des Addierers, der eigene Speicher oder einer der 4 Nachbarn ausgewählt.
Über den Row- oder Column-Highway werden 64 Bit-Worte transportiert, die je ein Bit für die Prozessoren in einer Reihe bzw. Spalte enthalten.

### 2.3.2 Der Speicher

Der Speicher des DAP ist insgesamt 8 MByte groß. Jedem Prozessor steht ein "Speicherfaden" der Länge 16 Kbit mit 1 Bit Breite zur Verfügung. Auf den Speicher haben sowohl der DAP als auch der Host Zugriff, was auch für die später beschriebene asynchrone Ein-/Ausgabe wichtig ist.
Der Datenaustausch zwischen Host und DAP erfolgt über diesen gemeinsamen Speicher.

### 2.3.3 Die Master Control Unit

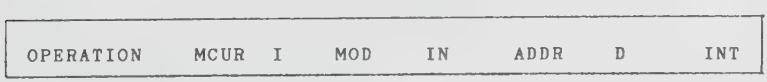
Die MCU hat im wesentlichen die Aufgabe der Adreßrechnung, der Befehlsdecodierung und der Verteilung der Befehle an die einzelnen Prozessoren. Da der DAP ein Single-Instruction-Multiple-Data-Rechner (SIMD) ist, wird ein Befehl an alle 4096 Prozessoren über sog. Highways verteilt.

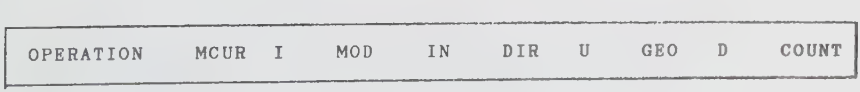


Die MCU hat als Besonderheit einen Instruction-Buffer, in dem 60 Befehle abgespeichert werden können. Dadurch werden bei Do-Schleifen Speicherzugriffe eingespart.

### 2.3.4 Befehlsformat

Jeder Assemblerbefehl umfaßt 32 Bit. Da die Befehle horizontal im Speicher abgelegt werden, enthält jede Speicherzeile 2 Befehle, die in einem Zyklus gelesen werden.
Im wesentlichen gibt es 2 unterschiedliche Befehlsformate:



Die einzelnen Felder bedeuten dabei folgendes:

| | |
|---|---|
| OPERATION | Operationscode |
| MCUR | MCU-Register 0 - 7 |
| I(NVERT) | Invert Operand |
| MOD | Adresse wird mit MCU-Register 0 - 7 modifiziert |
| IN(CREMENT) | Adresse wird um 1 erhöht |
| D(ECREMENT) | Adresse wird um 1 erniedrigt |
| ADDR | Adresse für Ebene |
| INT | Adresserweiterung für Spalten- und Zeilenadressierung |
| DIR(ECTION) | Richtung für Shifts |
| U(NUSED) | Nicht benutzt |
| GEO(METRY) | Ebene oder zyklische Geometrie bei Shifts |
| COUNT | Anzahl der Shifts |

## 2.4 DAP-Fortran

Für den DAP wurde eine eigene Sprache ( DAP-Fortran ) entwickelt.
Dem von seriellen Rechnern bekannten Fortran IV wurden einige Besonderheiten hinzugefügt:

1. Die neuen Datentypen Vektor und Matrix.
   Ein Vektor hat immer die Dimension 64, eine Matrix immer die Dimension 64 x 64.

2. Erweiterung der arithmetischen Grundoperationen und der Standardfunktionen SQRT,EXP,SIN usw. auf diese neuen Datentypen. SIN(X) bedeutet dann im Falle X = Matrix, daß der Sinus von jedem Matrixelement gleichzeitig berechnet wird.

3. Definition neuer Funktionen, die insbesondere das Arbeiten mit der speziellen Struktur des DAP erleichtern.
   Dazu gehören Schiebeoperation, Maskenbildung, Maximum-/ Minimumsuche,logische Operationen auf Masken, Summenbildung und Datenkonvertierung [FORT79].

Damit wurde erreicht, daß man Problemlösungen, die sich auf die Matrixstruktur des DAP abbilden lassen, sehr knapp und überschaubar programmieren kann.

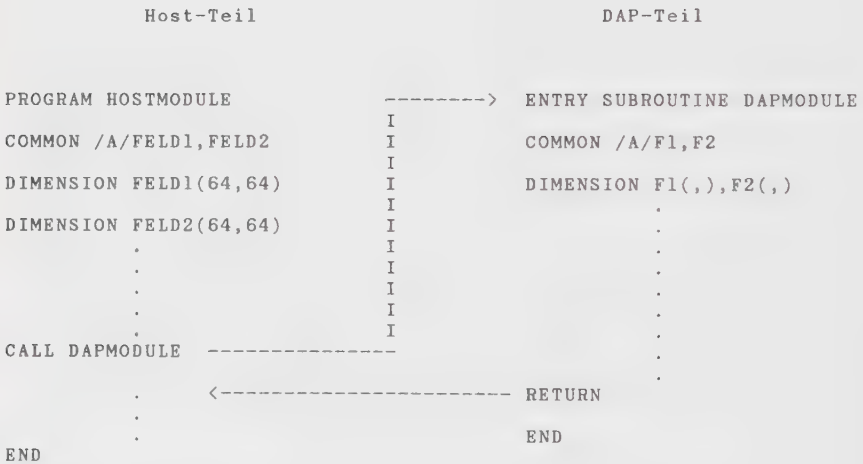
Um einen Einblick in die Sprache zu bekommen, sollen hier einige Beispiele exemplarisch angeführt werden:

```
DIMENSION A1(,),A2(,),A3(,)      - Vereinbarung von Matrizen
LOGICAL A3                       - der Dimension 64 x 64
A1=5.5                           - Jedes Element der Matrix
                                   A1 erhält den Wert 5.5
A1=A1+A2                         - Die Matrizen A1 und A2
                                   werden elementweise addiert
A2=SQRT(A1)                      - Es wird die Quadratwurzel
                                   von jedem Matrixelement
                                   berechnet
A1=MERGE(A1,A2,A3)               - Die Matrizen A1 und A2
                                   werden abhängig von der
                                   logischen Matrix A3 gemischt
S=SUM(A1)                        - Aufsummieren aller Elemente
                                   einer Matrix
S=MAXV(A2)                       - Bestimmen des Maximums
                                   aller Matrixelemente
A1=SHWC(A1)                      - Shiften der Matrix in west-
                                   liche Richtung
A1(A2.GE.A1)=A2                  - Die Wertzuweisung für die
                                   Elemente von A1 erfolgt nur
                                   dort, wo die Klammerbedingung
                                   wahr ist
A1=A1(,+)+A2(-,)                 - Vor Ausführung der Addition
                                   wird A1 um eine Stelle nach
                                   Westen und A2 um eine Stelle
                                   nach Süden geshiftet
```

## 2.5 Kommunikation Host-DAP

Das komplette Programm und alle Daten werden vom Host in den DAP-Speicher geladen. Anschließend wird der Host-Teil des Programms gestartet. Durch einen speziellen Subroutine-Aufruf wir vom Hostprogramm der DAP-Teil aktiviert. Dies bedeutet, daß die MCU mit der Startadresse des Befehlsbereichs versorgt und anschließend gestartet wird. Die Befehle sind 32 Bits lang und horizontal im Speicher abgelegt. Diese Anordnung entspricht der Abspeicherung von 32 Bit-Befehlen und -Daten des Host-Rechners.
Sie bedeutet, daß sich ein Befehl über den Speicher von 32 Prozessoren erstreckt. In der Befehlsholphase wird jeweils ein 64-Bit-Wort, das 2 Befehle enthält, über den sogenannten Column-Highway in einer Zykluszeit zur MCU übertragen. Anschließend werden die beiden Befehle in 2 Zykluszeiten ausgeführt.
Die Beendigung des DAP-Programmteils wird dem Host per Interrupt angezeigt, der daraufhin den pausierenden Hostteil reaktiviert.

Das folgende Beispiel verdeutlicht den Ablauf eines Programms:

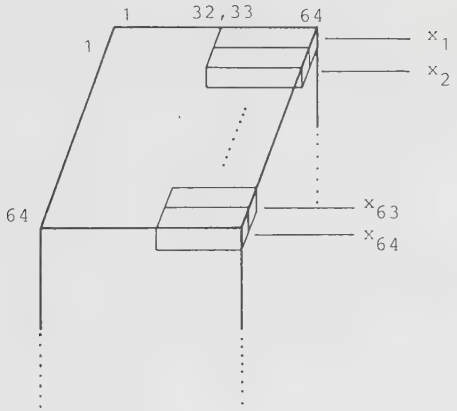



Die Kommunikation zwischen Host und DAP erfolgt über gemeinsame Common-Bereiche.

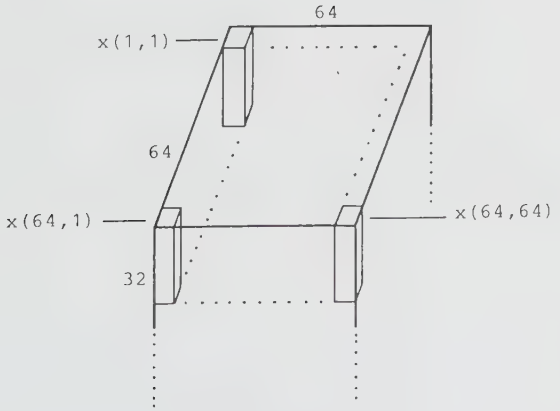
Dabei ist zu beachten, daß die Daten im Host anders abgelegt werden als im DAP. Die Daten müssen deshalb konvertiert werden.

Der DAP kennt zwei Ablagemodi:

### 2.5.1 Der Vektormodus




### 2.5.2 Der Matrixmodus



## 2.6 Arithmetik auf dem DAP

Besonders der Matrixmodus ist für eine schnelle Arithmetik geeignet. Da durch +,-,* oder / zu verknüpfende Daten im Speicher eines Prozessors liegen, ist kein Datenaustausch mit Nachbarprozessoren notwendig.
Bei derselben Operation im Vektormodus, bei der die zu verknüpfenden Daten über den Speicher von 32 Prozessoren verteilt sind, müssen Überträge an Nachbarprozessoren weitergegeben werden.
Die Überlegenheit von arithmetischen Operationen im Matrixmodus läßt sich am besten durch absolute Zahlen darstellen.
So dauert ein Addition im

- Vektormodus ( 64 Elemente ) $59.8 * 10^{-6}$ Sekunden
- Matrixmodus ( 4096 Elemente ) $157.3 * 10^{-6}$ Sekunden.

Das bedeutet, daß man in der Zeit, in der man im Matixmodus 4096 Elemente addieren kann, im Vektormodus nur etwa 192 Elemente addieren kann. Ähnlich sind die Verhältnisse bei Subtraktion, Multiplikation und Division.
Für die Berechnung der Standardfunktionen wurden neue Algorithmen ( sog. Bitalgorithmen ) implementiert.

Diese Bitalgorithmen berechnen aus dem Bitmuster des Arguments durch einfache Schiebeoperationen und Additionen das Ergebnis. Der Rechenzeitaufwand für Standardfunktionen ist nicht vergleichbar mit jenem bei konventionellen Rechenanlagen, da dort diese Funktionen meist mit einer Reihenentwicklung berechnet werden. So ist z.B. das Wurzelziehen am DAP schneller als eine Gleitpunktmultiplikation [MEGG62], [CHEN73], [GOST79], [REIN83].

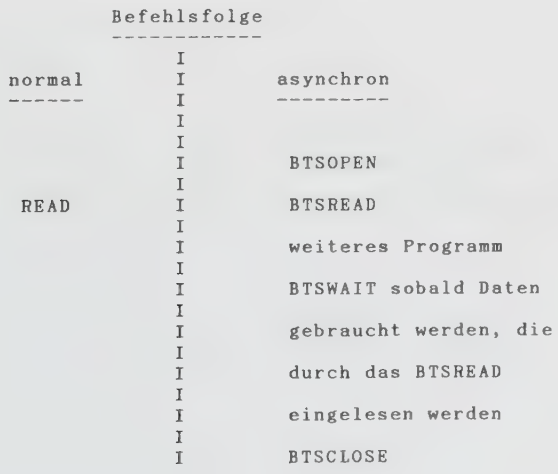
## 3. Asynchrone Ein-/Ausgabe am DAP

Trotz des relativ großen Arbeitsspeichers von 8 MByte gibt es auch im DAP bei der Bearbeitung großer Matrizen Probleme mit der Speichergröße. Da sowohl der DAP als auch der Host auf den gemeinsamen Speicher zugreifen können, liegt es nahe, zu versuchen, während des Rechenvorgangs im DAP nicht benötigte Daten auszulagern bzw. neue einzuschreiben. Diese Möglichkeit, die im weiteren als asynchrone Ein-/Ausgabe bezeichnet wird, ist bisher am DAP nicht angewendet worden.

Möglich wird dies durch unterschiedliche Zykluszeiten der Hardware-Komponenten. Der DAP hat eine Zykluszeit von 200 Nanosekunden, die Zykluszeit des Speichers ist kleiner als 100 ns. Dadurch ist es möglich, daß der Host vom DAP nicht genutzte Speicherzyklen selbst ausnutzt. Dies ist in der Literatur unter dem Begriff "cycle-stealing" bekannt.

Das Betriebssystem stellt hierzu ein asynchrones I/O-Interface BTS (Block-Transfer-System) zur Verfügung. Es enthält alle wesentlichen Befehle wie Open file, Close file, Read, Write, Wait, Query für die Dateibearbeitung und die Datenübertragung. Der Benutzer muß Puffer anlegen, in die die Daten übertragen werden. Sämtliche Dienstleistungen sind als Fortran-Subroutine-Aufrufe realisiert.

Während bei der normalen Ein-/Ausgabe in einem Fortran-Programm auf die Beendigung des READ/WRITE-Befehls gewartet wird und danach die Daten dem Programm sofort zur Verfügung stehen, wird bei der asynchronen Ein-/Ausgabe der Datentransfer nur angestoßen und anschließend sofort der nächste Befehl im Programm bearbeitet. Der Benutzer ist selbst dafür verantwortlich zu prüfen, ob die Übertragung der Daten, die er im Programm verwenden will, tatsächlich abgeschlossen ist. Andererseits hat dieses Verfahren den Vorteil, daß während eines Datentransfers das Programm weiterläuft und z.B. auf anderen Daten rechnen kann.
Stellt man normale und asynchrone Ein-/Ausgabe noch einmal gegenüber, so ergibt sich folgendes (Aufrufparameter sind der Übersichtlichkeit wegen weggelassen worden):

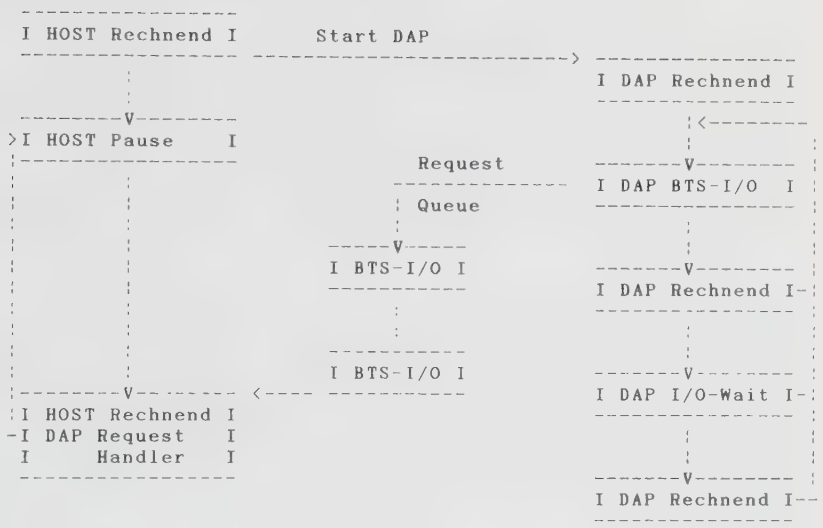


Der READ-Befehl im Normalfall enthält implizit das WAIT, da in dem betroffenen Programm erst fortgefahren wird, wenn der Transfer beendet ist.
Der Benutzer muß sich auch nicht um das Öffnen und Schließen von Dateien kümmern, wie dies im asynchronen Fall notwendig ist.
Im asynchronen Fall kann zwischen dem BTSREAD und dem BTSWAIT eine beliebige Anzahl von Befehlen stehen, die sofort abgearbeitet werden.
Sollen die gerade eingelesenen Daten Verwendung finden, muß sich der Benutzer mit BTSWAIT davon überzeugen, daß der Transfer abgeschlossen ist. Beim Aufruf von BTSWAIT wird solange gewartet, bis der abgefragte Datentransfer zu Ende ist. Die unmittelbare Folge der beiden Befehle BTSREAD und BTSWAIT entspricht dem normalen READ-Befehl in Fortranprogrammen.

### 3.1 Ablaufmodell 1

Unter Berücksichtigung der von ICL zur Verfügung gestellten Systemsoftware ergibt sich daraus folgendes Ablaufmodell:

Modell 1



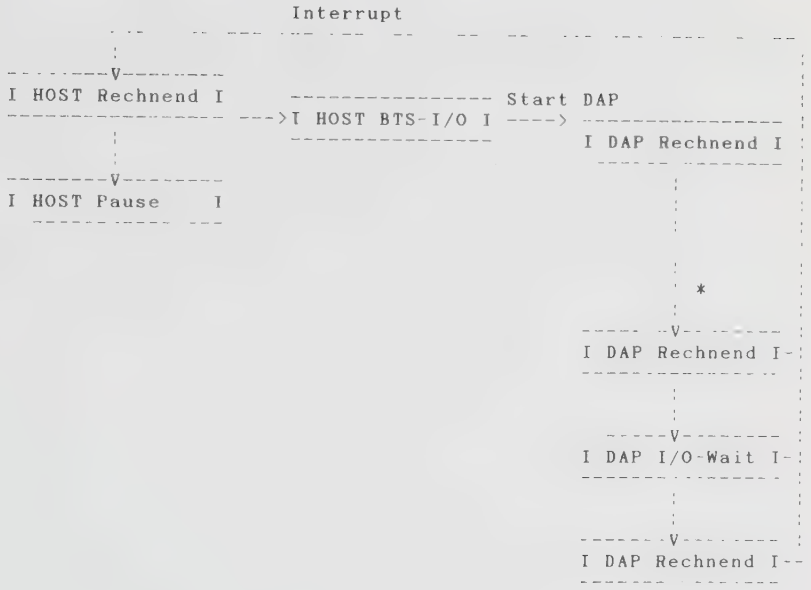

Der Host geht, nachdem er den DAP gestartet hat, für diesen Benutzer in Pause und bearbeitet andere Tasks.
Der DAP baut eine Request-Queue mit I/O-Aufträgen auf und weckt den Host zur Bearbeitung, sobald sich mindestens ein Element in der Warteschlange befindet. Der Host arbeitet die Warteschlange ab und startet entsprechende I/O-Aufträge.
In der Zwischenzeit rechnet der DAP unbehindert weiter. Die Synchronisation geschieht über eine Variable, in der die Nummer des zuletzt beendeten I/O-Auftrags abgelegt wird.

Messungen bei verschiedener Auslastung des Systems haben ergeben, daß der Weckvorgang für den Host sehr stark abhängig ist von der Auslastung des Hosts durch andere Benutzer.
Fuer identische Programmläufe lag die Weckzeit bei leerer Maschine bei ca. 0.002 s,bei voll ausgelastetem Host jedoch bei 0.83 s, was etwa einem Verhältnis von 1 : 400 entspricht.
Ebenso lastabhängig waren die Kanalzeiten und die damit verbundenen DAP-Wartezeiten auf neue Daten. Hier beträgt das Verhältnis etwa 1 : 10.
Während dieses Weckvorgangs und der anschließenden Datenübertragung geht der DAP in eine aktive Warteschleife. Das heißt, es wird ein Programmstück durchlaufen, in dem nur auf die Beendigung des Datentransfers gewartet wird. Der DAP verbraucht dadurch Rechenzeit und ist zusätzlich für andere Benutzer blockiert.

### 3.2 Ablaufmodell 2

Durch diese für den Benutzer nicht beeinflußbaren Umstände kann der Rechenzeitbedarf bzgl. des DAP bei Benutzung der asynchronen Ein-/Ausgabe nicht sinnvoll abgeschätzt werden. Deshalb mußte ein anderes Modell entwickelt werden, daß die Nachteile des 1. Modells beseitigt. Dieses 2. Modell wurde auf seine Eignung untersucht und diente dann, da das Ergebnis dieser Untersuchung positiv war, als Grundlage für die Implementierung am DAP.

Modell 2




Bei diesem Modell ist im wesentlichen der Host verantwortlich für den Start der asynchronen Ein-/Ausgabe. Erst wenn eine vom Algorithmus abhängige, bestimmte Anzahl von Ein-/Ausgabe-Aufträgen gestartet wurde, wird auch der DAP-Teil des Programms aktiviert. Der DAP kann sich an der asynchronen Ein-/Ausgabe derart beteiligen, daß bereits bei * und nicht erst durch den Host nicht mehr benötigte Daten asynchron ausgelagert werden.

Durch geschickte Wahl der Anzahl der Ein-/Ausgabeaufträge kann man die Wartezeit des DAP auf Daten auf null drücken.
Dies wird in einem späteren Abschnitt gezeigt.
Ist der DAP mit der Bearbeitung der Daten fertig, meldet er sich per Interrupt beim Host zurück. Dieser startet erneut weitere Ein-/Ausgabeaufträge bevor er dann wieder den DAP aktiviert, der in der Zwischenzeit für einen anderen Benutzer tätig war.
Auf diese Art und Weise bekommt man unabhängig von der Last im Host bei identischen Läufen auch immer gleiche DAP-CPU-Zeiten.

## 3.3 Vergleich der Modelle

Um den synchronen Ablauf und die Modelle 1 und 2 vergleichen zu können, muß man zunächst den zeitlichen Ablauf betrachten.

Synchroner ( "normaler" ) Ablauf

Im Normalfall wechseln sich der Host und der DAP in der Bearbeitung eines Jobs ab. Zunächst wird der Host gestartet, der eventuell notwendige Eingaben macht und anschließend den DAP aktiviert. Ist der DAP mit der Bearbeitung fertig, meldet er sich beim Host per Interrupt zurück. Hier werden wieder Ein- und Ausgaben abgearbeitet, anschließend wird der DAP erneut gestartet.
Dieser Vorgang kann sich beliebig oft wiederholen.
Wichtig ist, daß Host und DAP nie gleichzeitig an der Bearbeitung eines Jobs beteiligt sind.

Modell 1

Wie im Normalfall wird der Host gestartet und dieser initialisiert anschließend den DAP. Der DAP setzt bei Punkt 1 asynchrone Ein-/Ausgabeaufträge ab, die vom Host ausgeführt werden. Wie bereits begründet wurde, kann es im DAP je nach Auslastung des Host zu größeren Wartezeiten W kommen. Dabei gibt der DAP die Regie nicht ab und durchläuft eine Warteschleife bis der entsprechende Ein-/Ausgabeauftrag beendet ist. Diese Warteschleifendurchläufe werden dem Benutzer voll als Rechenzeit berechnet.

Modell 2

Nach dem Starten des Host initialisiert dieser zunächst Ein-/Ausgabeaufträge und startet danach den DAP. Auch hier kann es zu Wartezeiten W kommen. Wie später gezeigt wird, kann man diese Wartezeiten durch geeignete Maßnahmen eliminieren oder sie zumindest vernachlässigbar klein machen.
Hier ist gegenüber dem synchronen Fall ein zeitliches Nebeneinander von Host und DAP bei der Bearbeitung eines Auftrags sinnvoll.

Das auf S.34 erwähnte "cycle-stealing" behindert die eigentliche Arbeit des DAP in keiner Weise. Um dies zu belegen, wurden einige Meßreihen gestartet, die sich am besten durch das folgende FORTRAN-Programm illustrieren lassen. Die einzelnen Parameter werden nicht näher erläutert. Sie können dem BTS-Handbuch entnommen werden.

Host-Programm:

```
      PROGRAM HOSTMUL
      COMMON /U/IDATA/V/N
      DIMENSION IDATA(64,64)
      CALL BTSSETUP(3)
C     INITIALISIEREN BTS-SYSTEM (BLOCK-TRANSFER-SYSTEM)
      CALL BTSOPEN(10,1,16384,3,0,16384,0,0)
C     OEFFNEN EINER DATEI MIT KANALNUMMER 10 ZUM TRANSPORT
C     VON 4096 ELEMENTEN ZU 32 BIT
      READ(5,2) N
2     FORMAT(I4)
C     EINLESEN DER LAUFSCHLEIFENGRENZE
      DO 1 K=1,20
      CALL BTSREAD(10,IDATA,1,0,ID1)
C     ABSETZEN EINES ASYNCHRONEN EINGABEAUFTRAGS
      CALL DAPMUL
C     AUFRUF EINES DAP-PROGRAMMS
      CALL BTSWAIT(10,ID1,IR)
C     ABFRAGE OB EINGABEAUFTRAG BEENDET
1     CONTINUE
      CALL BTSCLOSE(10)
C     DATEI SCHLIESSEN
      END
```

DAP-Teil

```
      ENTRY SUBROUTINE DAPMUL
      COMMON /U/IDATA/V/N
      DIMENSION IDATA(,),DATA(,)
      CALL CONVFSI(N)
C     KONVERTIERUNG VON N WEGEN UNTERSCHIEDLICHER
C     DATENABLAGE IM HOST UND IM DAP
      DATA=1.
C     ALLEN ELEMENTEN DER 64x64-MATRIX WIRD DER WERT 1 ZUGEWIESEN
      DO 1 K=1,N
1     DATA=DATA*1.
      RETURN
      END
```

Meßreihe 1:

Die DAP-Programme führen in einer Laufschleife n Multiplikationen aus. Dabei wird n in 7 Jobs von 2000 bis 8000 in 1000-ter Schritten erhöht. Alle Jobs liefen einmal ohne und einmal mit gleichzeitigem asynchronen Einlesen einer Matrix.

Meßreihe 2:

Wie bei 1, nur wurden Additionen ausgeführt.

Meßreihe 3:

Wie bei 1, nur wurden Schiebeoperationen ausgeführt.

Meßreihen 4,5,6:

Wie Meßreihen 1,2,3 mit der Anpassung von n derart, daß asynchrone Eingabe und DAP-Rechenvorgang gleichzeitig beendet waren.

Alle Meßreihen wurden mehrfach gestartet. Die im Accounting angegebenen DAP-CPU-Zeiten waren für identische Läufe immer die gleichen.

Dies bedeutet als erstes wichtiges Ergebnis:

> Asynchrone Ein-/Ausgabe behindert die Arbeit des DAP in keiner Weise. Dies ist insbesondere auch unabhängig von der momentanen Last am Host.

Zweites Ergebnis:

> Die Ablaufdauer der asynchronen Ein-/Ausgabe ist stark abhängig von den gerade im DAP ablaufenden Befehlen.

Das asynchrone Einlesen einer Matrix mit 4096 Elementen und 32 Bit Wortlängen nimmt folgende Zeiten in Anspruch:

| Im DAP laufende Befehle | Verstrichene DAP-CPU-Zeit |
|---|---|
| Integer-Additionen ( Variable + Konstante ) | 158 ms |
| Real-Multiplikationen | 193 ms |
| Schiebeoperationen | 208 ms |

Die unterschiedlichen Zeiten für das Einlesen gleichgroßer Matrizen lassen sich durch die unterschiedlich hohen Speicherzugriffsraten bei unterschiedlichen Befehlen erklären.
Bei den Schiebeoperationen dauert das asynchrone Einlesen wesentlich länger als z.B. bei der Integer-Addition von einer Variablen und einer Konstanten. Dies ist aber leicht einsehbar, wenn man den Assembler-Code betrachtet.
Bei der Integer-Addition ( Variable + Konstante ) wird im wesentlichen eine Laufschleife durchlaufen, die aus den Befehlen QB ( Register --> Register ) und SICPCQS ( Register --> Speicher) besteht. Das bedeutet, das mit jedem 2. Befehl auf den Speicher zugegriffen wird.

Bei Schiebeoperationen wird eine Laufschleife durchlaufen, die aus den Befehlen QS ( Speicher --> Register ), QQ ( Register --> Register ) und SQ ( Register --> Speicher ) besteht. Damit greifen 2 von 3 Befehlen auf den Speicher zu. Rein theoretisch müssen sich deshalb die benötigten Zeiten zum Einlesen wie das Verhältnis der freien Speicherzyklen verhalten. Das rechnerische Ergebnis 208 : 158 = 1.32 kommt dem theoretischen Wert 1/2 : 1/3 = 1.5 durchaus nahe.

# 4. Das Invertierungsverfahren von Sherman-Morrison und seine Modifizierung für den DAP

In der Literatur sind zahlreiche Verfahren zur Invertierung von Matrizen bekannt. Der Rechenaufwand vergrößert sich jedoch mit zunehmender Dimension der Matrizen sehr stark, da die Verfahren meist von der Ordnung $O(n^3)$ sind, wie z.B. auch das Gaußsche Eliminationsverfahren.
Dadurch treten bei der Invertierung großer Matrizen erhebliche Rechenzeitprobleme auf.

## 4.1 Invertierungsverfahren

Allgemein unterscheidet man zur Lösung von Gleichungssystemen bzw. zur Invertierung von Matrizen im wesentlichen 2 Gruppen von Verfahren:

### 4.1.1 Direkte Verfahren

Direkte Verfahren berechnen nach endlich vielen Schritten (meist n Schritten, mit n = dim (A)) die Lösung zu $Ax = b$ bzw. $A^{-1}$.
Könnte man rundungsfehlerfrei rechnen, wie z.B. in der Residuen-Arithmetik, dann wäre das Ergebnis die exakte Lösung.
Zu den direkten Verfahren gehören z.B. die bereits erwähnte Gauß-Elimination, die Cholesky-Zerlegung, die Faktorisierung nach Crout und das Verfahren von Sherman-Morrison [WEST68].

### 4.1.2 Iterative Verfahren

Iterative Verfahren berechnen zu $Ax = b$ eine Näherungslösung $x^{(k)}$. Dabei ist für die Berechnung stets ein Abbruchkriterium notwendig, z.B.

$$\| x^{(k)} - x^{(k-1)} \| = \max_{1 \leq i \leq n} | x_i^{(k)} - x_i^{(k-1)} | < e.$$

Das heißt, die Berechnung wird abgebrochen, wenn sich der Lösungsvektor in jeder Komponente $x_i$ nur noch um maximal e von der vorhergehenden Näherungslösung unterscheidet.
Analoges gilt für die iterative Berechnung von $A^{-1}$ durch eine Folge von $A^{-1}_{(k)}$.

Zu diesen Verfahren gehören das Gauß-Seidel-, das Jacobi-Verfahren und die Relaxationsverfahren aber auch Monte-Carlo-Methoden und Block-iterative Verfahren. Die Anzahl k der Rechenschritte ist hierbei nicht abhängig von der Dimension der Matrix, sondern von der Konvergenz des Verfahrens.

Dies bedeutet, daß insbesondere k wesentlich größer als n werden kann.

### 4.1.3 Vorteile des ausgewählten Verfahrens

Sowohl die direkten Verfahren als auch die iterativen Verfahren haben den Nachteil, daß pro Berechnungsschritt die gesamte Matrix geändert werden muß.

Wie später gezeigt wird, kommt das ausgewählte Verfahren mit geeigneten Modifikationen im Mittel mit Änderungen in n/2 Spalten der Matrix aus.
Es hat weiterhin den Vorteil, daß die Berechnung von Inversen ähnlicher Matrizen sehr einfach ist.

Beispiel:

Sei A eine Matrix für ein lineares Gleichungssystem und $A^{-1}$ bekannt.
Ändert sich in diesem Gleichungssystem eine Gleichung, so muß zu diesem neuen A' nicht ein vollkommen neues $A'^{-1}$ berechnet werden. $A'^{-1}$ läßt sich mit dem Verfahren von Sherman-Morrison in einem Berechnungsschritt aus $A^{-1}$ gewinnen[SHM050].

## 4.2 Große Matrizen

Probleme in den Ingenieurwissenschaften wie z.B. Strömungsberechnungen ( Finite-Elemente-Verfahren) oder Berechnungen für Kernkraftwerke ( Lösung der Navier-Stokes-Gleichung in 3 Dimensionen) lassen die Größe der zu lösenden Gleichungssysteme sehr stark ansteigen. Matrizen der Dimension 10000 x 10000 und größer sind dabei keine Seltenheit.

Solche Gleichungssysteme können auf herkömmlichen Rechnern, wenn überhaupt, nur mit großem Rechenzeitaufwand gelöst werden.

Neben der hohen Rechenzeit ( im CPU-Tage-Bereich ) geht hier noch erschwerend ein, daß der Arbeitsspeicher nicht ausreicht, die gesamten Daten zu speichern. Dadurch müssen Daten ständig vom Hintergrundspeicher nachgeladen bzw. auf diesen ausgelagert werden. Die dabei auftretenden Kanalbelegungszeiten übersteigen die reinen Rechenzeiten um ein Vielfaches.

Die hohe Leistungsfähigkeit des DAP, die die benötigte CPU-Zeit gegenüber herkömmlichen Rechner ( z.B. CYBER 173 ) um eine Zeitgrößenordnung verbessert ( d.h. statt CPU-Stunden werden nur noch CPU-Minuten gebraucht ), gestattet es, die Lösung von großen Gleichungssystemen in Angriff zu nehmen.

Da aber auch hier der Arbeitsspeicher nur endlich groß ist, muß man versuchen, ein Lösungsverfahren zu finden, das pro Lösungsschritt nur einen Teil der Daten benötigt.

Ein wirksamer Mechanismus für das effiziente Aus- und Nachladen von Daten wurde in Kapitel 3 vorgestellt.

## 4.3 Das Verfahren von Sherman-Morrison

Der Algorithmus wurde 1949 veröffentlicht als es noch keine Parallelrechner gab. Er eignet sich mit einigen Modifikationen sehr gut für den DAP, auch und insbesondere unter Berücksichtigung der asynchronen Ein-/Ausgabe.

Gesucht wird das Inverse $B^{-1}$ einer Matrix B mit dim(B)=n.
Man geht bei diesem Verfahren davon aus, daß es eine Matrix A mit dim(A)=n gibt, deren Inverses $A^{-1}$ bekannt ist.
Ausgehend von A und $A^{-1}$ berechnet man in n Schritten B und $B^{-1}$.

Der k-te Schritt sieht dabei folgendermaßen aus:

1. In der Matrix A wird die k-te Spalte durch die k-te Spalte aus der Matrix B ersetzt und der Hilfsvektor V mit

$$v_i = \sum_{j=1}^{n} a^{-1}_{ij} \, b_{jk} \qquad \text{für} \quad 1 \leq i \leq n$$

berechnet.

2. Berechnung des Inversen

a) Berechnung der k-ten Zeile

$$b^{-1}_{kj} = a^{-1}_{kj} / v_k \qquad \text{für} \quad 1 \leq j \leq n$$

b) Berechnung der übrigen Zeilen

$$b^{-1}_{ij} = a^{-1}_{ij} - v_i b^{-1}_{kj} \qquad \text{für } 1 \leq i,j \leq n,\ i \neq k$$

3. $A^{-1} = B^{-1}$

Nach n Schritten ist dann aus der Matrix A die Matrix B entstanden und aus $A^{-1}$ die gesuchte Matrix $B^{-1}$.

Der Beweis für die Richtigkeit dieser Berechnungsvorschrift, der in der Literatur nicht angegeben ist, wird in folgender Weise erbracht:

Es muß gelten:

I) $$\sum_{l=1}^{n} b^{-1}_{kl}\, b_{lk} = 1$$

II) $$\sum_{l=1}^{n} b^{-1}_{rl}\, b_{lk} = 0 \qquad 1 \leq r \leq n,\ r \neq k$$

III) $$\sum_{l=1}^{n} b^{-1}_{rl}\, b_{lr} = \sum_{l=1}^{n} a^{-1}_{rl}\, b_{lr} \qquad 1 \leq r \leq n,\ r \neq k$$

Der Beweis erfolgt durch Einsetzen von 1. und 2. der Berechnungsvorschrift.

I)
$$\sum_{l=1}^{n} b^{-1}_{kl}\, b_{lk} = \sum_{l=1}^{n} \frac{a^{-1}_{kl}}{v_k}\, b_{lk} =$$

$$\sum_{l=1}^{n} \frac{a^{-1}_{kl}\, b_{lk}}{\sum_{j=1}^{n} a^{-1}_{kj}\, b_{jk}} = \frac{\sum_{l=1}^{n} a^{-1}_{kl}\, b_{lk}}{\sum_{j=1}^{n} a^{-1}_{kj}\, b_{jk}} = 1$$

II)
$$\sum_{l=1}^{n} b^{-1}_{rl}\, b_{lk} = \sum_{l=1}^{n} \left( a^{-1}_{rl} - v_r\, b^{-1}_{rkl} \right) b_{lk} =$$

$$\sum_{l=1}^{n} a^{-1}_{rl}\, b_{lk} - v_r \sum_{l=1}^{n} b^{-1}_{kl}\, b_{lk} = \sum_{l=1}^{n} a^{-1}_{rl}\, b_{lk} - v_r =$$

$$\sum_{l=1}^{n} a^{-1}_{rl}\, b_{lk} - \sum_{l=1}^{n} a^{-1}_{rl}\, b_{lk} = 0$$

III)

$$\sum_{l=1}^{n} b^{-1}_{rl} b_{lr} = \sum_{l=1}^{n} (a^{-1}_{rl} - v_r b^{-1}_{kl}) b_{lr} =$$

$$\sum_{l=1}^{n} a^{-1}_{rl} b_{lr} - \sum_{l=1}^{n} v_r b_{lr} b^{-1}_{kl} =$$

$$\sum_{l=1}^{n} a^{-1}_{rl} b_{lr} - \frac{v_r}{v_k} \sum_{l=1}^{n} b_{lr} a^{-1}_{kl} = \sum_{l=1}^{n} a^{-1}_{rl} b_{lr}$$

## 4.4 Modifikation des Lösungsverfahrens

Das Verfahren von Sherman-Morrison eignet sich in seiner ursprünglichen Form nicht für den Einsatz am DAP.
Auch bei diesem Verfahren müssen pro Berechnungsschritt alle Spalten geändert werden. Da dies zu aufwendig wäre und sehr viele Daten hin- und hergeladen werden müßten, muß das Verfahren für den DAP modifiziert werden.

Um den bei großen Matrizen erheblichen Speicheraufwand für A und $A^{-1}$ zu umgehen und den Rechenaufwand zu reduzieren, wird eine 1. Modifikation am Verfahren vorgenommen:

Man geht nicht von einer beliebigen Matrix A aus, sondern wählt

$$A = A^{-1} = I.$$

Da die Einheitsmatrix I nur in der Diagonalen Eins ist und sonst Null, ergibt sich, daß im k-ten Schritt nur die ersten k Spalten neu berechnet werden müssen.

Für den k-ten Schritt ergibt sich jetzt folgende neue Berechnungsvorschrift:

1. Berechnung des Hilfsvektors V mit

$$v_i = \sum_{j=1}^{k-1} a^{-1}_{ij} \, b_{jk} + \begin{cases} 0 & \text{für } i < k \\ b_{ik} & \text{für } i \geq k \end{cases}$$

2. Berechnung der Inversen

$$b_{ij}^{-1} = \begin{cases} 1/v_k & i=j=k \\ a_{ij}^{-1}/v_k & i=k \\ -v_i/v_k & j=k,\ i \neq k \\ a_{ij}^{-1} - a_{kj}^{-1} v_i/v_k & i \neq k,\ j \neq k \end{cases} \quad \left.\begin{array}{l} \\ \\ \\ \\ \end{array}\right\} \; 1 \leq i,j \leq n$$

3. $A^{-1} = B^{-1}$

Auch hier erhält man nach n Schritten die gesuchte Matrix $B^{-1}$.

Es ist wieder I, II und III durch Einsetzen zu beweisen.

I)

$$\sum_{l=1}^{n} b_{kl}^{-1} b_{lk} = \sum_{l=1}^{k-1} \frac{a_{kl}^{-1}}{v_k} b_{lk} + \frac{b_{kk}}{v_k} =$$

$$\frac{\sum_{l=1}^{k-1} a_{kl}^{-1} b_{lk}}{\sum_{j=1}^{k-1} a_{kj}^{-1} b_{jk} + b_{kk}} + \frac{b_{kk}}{\sum_{j=1}^{k-1} a_{kj}^{-1} b_{jk} + b_{kk}} = 1$$

II)

Fall 1: $r < k$

$$\sum_{l=1}^{n} b^{-1}_{rl} b_{lk} = \sum_{l=1}^{k-1} \left( a^{-1}_{rl} - a^{-1}_{kl} \frac{v_r}{v_k} \right) b_{lk} - \frac{v_r}{v_k} b_{kk} =$$

$$\sum_{l=1}^{k-1} a^{-1}_{rl} b_{lk} - \frac{v_r \left( \sum_{l=1}^{k-1} a^{-1}_{kl} b_{lk} + b_{kk} \right)}{v_k} =$$

$$\sum_{l=1}^{k-1} a^{-1}_{lr} b_{lk} - \frac{\sum_{l=1}^{k-1} a^{-1}_{rl} b_{lk} \left( \sum_{l=1}^{k-1} a^{-1}_{kl} b_{lk} + b_{kk} \right)}{\sum_{l=1}^{k-1} a^{-1}_{kl} b_{lk} + b_{kk}} = 0$$

Fall 2: $r > k$

$$\sum_{l=1}^{n} b^{-1}_{rl} b_{lk} = \sum_{l=1}^{k-1} \left( a^{-1}_{rl} - a^{-1}_{kl} \frac{v_r}{v_k} \right) b_{lk} - \frac{v_r}{v_k} b_{kk} + b_{rk} =$$

$$\sum_{l=1}^{k-1} a^{-1}_{rl} b_{lk} + b_{rk} - \frac{v_r \left( \sum_{l=1}^{k-1} a^{-1}_{kl} b_{lk} + b_{kk} \right)}{v_k} =$$

$$\sum_{l=1}^{k-1} a_{rl}^{-1} b_{lk} + b_{rk} - \frac{\left(\sum_{l=1}^{k-1} a_{rl}^{-1} b_{lk} + b_{rk}\right)\left(\sum_{l=1}^{k-1} a_{kl}^{-1} b_{lk} + b_{kk}\right)}{\sum_{l=1}^{k-1} a_{kl}^{-1} b_{lk} + a_{kk}^{-1} b_{kk}} = 0$$

III)

Fall 1: $r < k$

$$\sum_{l=1}^{n} b_{rl}^{-1} b_{lr} = \sum_{l=1}^{k-1} \left(a_{rl}^{-1} - a_{kl}^{-1} \frac{v_r}{v_k}\right) b_{lr} - \frac{v_r}{v_k} b_{kr} =$$

$$\sum_{l=1}^{k-1} a_{rl}^{-1} b_{lr} - \frac{v_r}{v_k} \left(\sum_{l=1}^{k-1} a_{kl}^{-1} b_{lr} + b_{kr}\right) =$$

$$\sum_{l=1}^{k-1} a_{rl}^{-1} b_{lr} - \frac{\sum_{l=1}^{k-1} a_{rl}^{-1} b_{lk} \left(\sum_{l=1}^{k-1} a_{kl}^{-1} b_{lr} + b_{kr}\right)}{\sum_{l=1}^{k-1} a_{kl}^{-1} b_{lk} + b_{kk}} =$$

$$\sum_{l=1}^{k-1} a_{rl}^{-1} b_{lr} = \sum_{l=1}^{n} a_{rl}^{-1} b_{lr}$$

Fall 2: $r > k$

$$\sum_{l=1}^{n} b_{rl}^{-1} b_{lr} = 1 = \sum_{l=1}^{n} a_{rl}^{-1} b_{lr}$$

Der Rechenaufwand ist nach dieser Modifikation nur noch halb so groß wie vorher.
Eine exakte Abschätzung des Rechenaufwands wird später vorgenommen.

Bisher wurde stillschweigend davon ausgegangen, daß die $v_k$ alle ungleich 0 sind. Dies ist normalerweise nicht der Fall.

Als 2. Modifikation wird deshalb die Pivotisierung eingeführt.

Sei $v_p = \max_{1 \le i \le n} \{ |v_i| , \text{i-te Spalte wurde noch nicht verwendet} \}$.

Ist im k-ten Schritt $p \neq k$, so wird die Inverse $B^{-1}$ nicht spaltenweise fortlaufend aufgebaut.
Aus den beiden ersten Zeilen der Berechnungsvorschrift für das Inverse sieht man dann sofort, daß das Inverse zeilenpermutiert ist.

Diese Permutationen kann man sich aber leicht in einem Vektor abspeichern.

Dazu werden 2 Vektoren der Länge n eingeführt. Ein logischer Vektor PIV und ein Integer-Vektor IPIV. Zu Beginn der Berechnung haben alle Elemente von PIV den Wert TRUE und alle Elemente von IPIV den Wert 0.

PIV (i) = TRUE bedeutet, daß die i-te Spalte noch nicht verwendet wurde.

IPIV (i) = k bedeutet, daß die k-te Zeile von $B^{-1}$ tatsächlich als i-te Zeile abgelegt wurde.

Nach dieser Modifikation ergibt sich folgende neue Berechnungsformel für den k-ten Schritt:

1. Hilfsvektor V mit

$$v_i = \sum_{j=1}^{n} a_{ij}^{-1} b_{jk} + \begin{cases} 0 & PIV(i) = FALSE \\ b_{ik} & PIV(i) = TRUE \end{cases}$$

$PIV(j)=FALSE$

2. Pivotelement

$$v_p = \max_{1 \le i \le n} |v_i|$$

$PIV(i)=TRUE$

3. Berechnung des Inversen

$$b_{ij}^{-1} = \begin{cases} 1/v_p & i=j=p \\ a_{ij}^{-1}/v_p & i=p,\ 1 \le j \le n,\ PIV(j)=FALSE \\ -v_i/v_p & j=p \\ a_{ij}^{-1} - a_{pj}^{-1} v_i/v_p & 1 \le j \le n,\ PIV(j)=FALSE \end{cases} \quad \left.\begin{array}{l} \\ \\ \\ \\ \end{array}\right\} \; 1 \le i \le n,\ i \neq p$$

4. $PIV(p) = FALSE$

$IPIV(p) = k$

$A^{-1} = B^{-1}$

Damit wurde das ursprüngliche Verfahren soweit modifiziert, daß es sich für den DAP eignet. Um eine gute Verarbeitung am DAP zu gewährleisten, müssen die sehr großen Matrizen in geeigneter Weise zerlegt und auf die Struktur des DAP abgebildet werden. Diese Zerlegung und die zugehörige Abbildung in den DAP-Speicher wird im nächsten Kapitel behandelt.

## 5. Die Abbildungsfunktion CRACKING

Der DAP mit seinen 64 x 64 Prozessoren eignet sich bestens für die Bearbeitung von Matrizen mit einer Dimension $\leq$ 64.
Dies bedeutet jedoch nicht, daß größere Matrizen nicht bearbeitet werden können. Man muß nur versuchen, größere Matrizen zu zerlegen und in geeigneter Weise auf den DAP-Speicher abzubilden. In der DAP-Literatur [WEBB80] sind dazu zwei Verfahren bekannt, die bisher jedoch nur auf Matrizen der Größe 128 x 128 angewendet wurden.

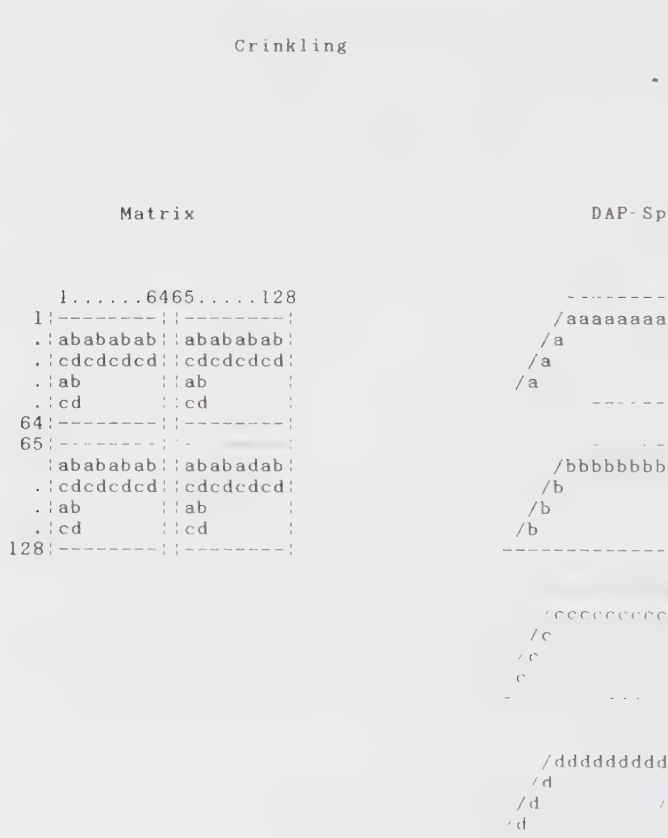
### 5.1 Das Crinkling

Hier wird die Matrix gewissermaßen zusammengedrückt und benachbarte Elemente werden im Speicher eines Prozessors abgelegt. Diese Abbildung ist immer dann sinnvoll, wenn in die Berechnung Nachbarschaftsbeziehungen einfließen, da hier benachbarte Elemente im Speicher eines Prozessors abgelegt sind und bei deren mathematischer Verknüpfung keine Prozessorkommunikation stattfinden muß.

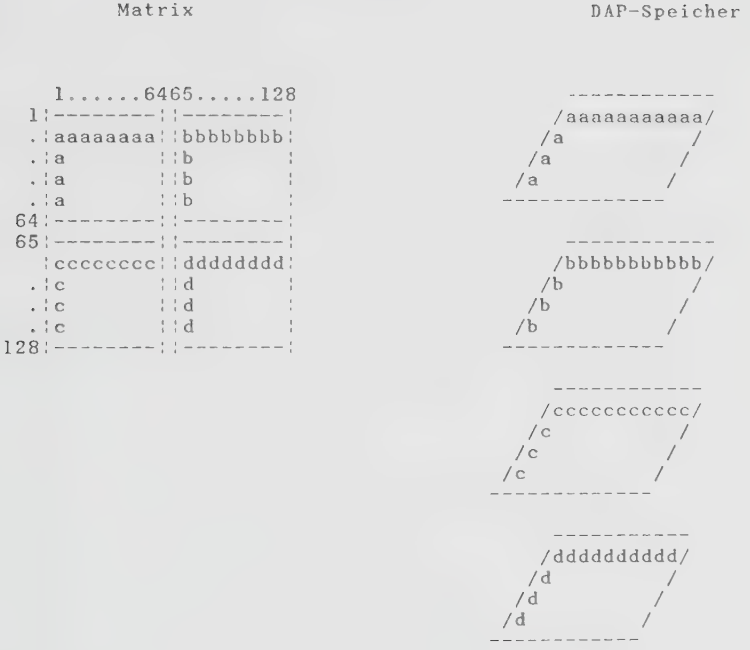
### 5.2 Das Slicing oder Partitioning

Hier wird die Matrix in Teile der Größe 64 x 64 zerschnitten. Sich in den einzelnen Teilen entsprechende Elemente werden im Speicher eines Prozessors abgelegt. Dies hat den Vorteil einer einfachen Indizierung.

Beide Verfahren werden in den beiden folgenden Bildern am Beispiel einer 128 x 128 - Matrix nochmals verdeutlicht:



Slicing


Matrix
DAP-Speicher
1.....6465.....128
1
aaaaaaaa
bbbbbbbb
a
b
64
65
cccccccc
dddddddd
c
d
128

## 5.3 Cracking

Da beim Verfahren von Sherman-Morrison Nachbarschaftsbeziehungen keine Rolle spielen, scheidet das Crinkling als Zerlegungsverfahren aus.
Erweitert man das Slicing auf beliebig große Matrizen und definiert eine Abbildungsfunktion, die die Matrizen in eine für das Invertierungsverfahren geeignete Art und Weise aufteilt, so erhält man eine neue Zerlegungsmöglichkeit, die mit Cracking bezeichnet wurde.

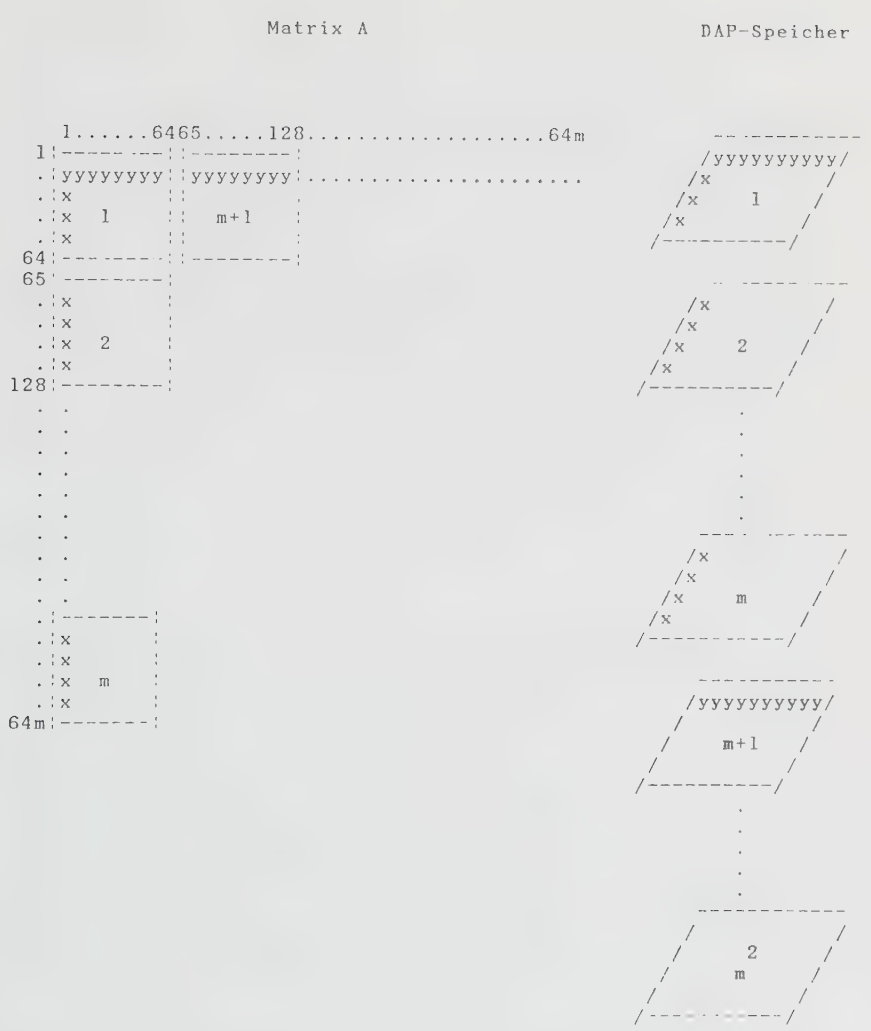
Sei A eine n x n - Matrix mit n = 64 m.
Sei f eine Abbildung von Matrixelementen in den DAP-Speicher

$$a_{ij} \xrightarrow{\ f\ } \text{DAP}\ a_{imod64,jmod64,e}$$

mit

$$e = m\,(\lceil j/64 \rceil - 1) + \lceil i/64 \rceil - 1 + 1$$

Diese Funktion f bewirkt, daß ein Element der Matrix A auf einen Speicherplatz der Ebene e abgebildet wird. Innerhalb dieser Ebene erhält das Element $a_{ij}$ dann den Platz, der sich ergibt, wenn man die Indizes i und j modulo 64 nimmt.
Die Abbildung läßt sich durch das folgendes Bild veranschaulichen.
Dabei wird die Matrix in Quadrate der Größe 64 x 64 zerlegt.
Sollte die Dimension der Matrix kein Vielfaches von 64 sein, dann wird im Berechnungsverfahren an den Rändern eine Maskenbildung vorgenommen.


Matrix A
DAP-Speicher
1.....6465.....128.....64m
1
yyyyyyyy
x
x 1
x
64
65
x
x
x 2
x
128
x
x
x m
x
64m
yyyyyyyy
m+1
yyyyyyyyyy
1
2
m
m+1
2
m

Ergebnis:

Mit diesem Cracking erreicht man, daß eine Spalte x in m aufeinander folgenden Ebenen und eine Zeile y in m Ebenen mit jeweils Abstand m abgelegt werden.
Damit hat man eine klare Anordnung der Matrix sowohl für das Adressieren einzelner Spalten als auch einzelner Zeilen.
Man kann jetzt gezielt jeweils eine Menge von 64 Spalten oder 64 Zeilen nachladen bzw. auslagern.

## 5.4 Eine weitere Modifikation des Invertierungsverfahrens

Um sowohl die Anzahl der Rechenoperationen als auch die Anzahl der Ein-/Auslagerungen auf der anderen Seite zu begrenzen, führt man eine 3. Modifikation, die eingeschränkte Pivotsuche, ein.

Dieser Modifikation liegt folgender Gedanke zugrunde:
Bei einer Pivotsuche über alle Elemente des Hilfvektors V ergibt sich eine statistische Verteilung der verwendeten Spalten.
Das kann im Extremfall dazu führen, daß jeweils nur eine aus 64 Spalten verwendet wurde. Nur diese eine Spalte muß im Berechnungsvorgang verändert werden, die 63 anderen Spalten bleiben unverändert. Dies führt dazu, daß der DAP nur zu 1/64 ausgelastet ist.

Um dies zu vermeiden, führt man eine Teilpivotsuche durch.
Im k-ten Schritt macht man eine Pivotsuche nur über die Elemente $v_i$ von V mit $1 \leq i \leq k + 64$ .
Diese Teilpivotsuche hat den Vorteil, daß die verwendeten Spalten gewissermaßen dicht liegen im Bereich von 1 bis k+64 und man nur jeden 64. Berechnungsschritt eine weiteren Block von 64 Spalten hinzunehmen muß. Alle Spalten, die sich nicht in diesem Bereich befinden, bleiben unverändert und können ausgelagert bleiben.

Dies geht natürlich nur so lange gut, wie es tatsächlich ein $v_i \neq 0$ für $1 \leq i \leq k+64$ gibt.
Ist dies nicht der Fall, so muß man für diesen einen Berechnungsschritt auf die volle Pivotsuche zurückgehen.

Diese Teilpivotsuche kann bei schlecht konditionierten Matrizen dazu führen, daß das Verfahren instabil wird. Dies ist dann der Fall, wenn das gefundene Teilpivotelement in seinem Wert stark vom Gesamtpivotelement abweicht.
Bei solchen Matrizen ist es deshalb wichtig, zunächst das Maximum über alle Elemente des Hilfsvektors zu ermitteln. Dies ist am DAP auf eine sehr schnelle Art und Weise möglich.
Anschließend bildet man den Quotienten $p/\max_i |v_i|$.

Ist dieser Quotient größer als eine vorgegebene Schranke e, so kann man mit dem Teilpivotelement p weiterrechnen. Ist dies nicht der Fall, muß man für diesen Iterationschritt auf die Vollpivotsuche zurückgehen.

Ergebnis:

Eine Teilpivotsuche ergibt eine erhebliche Reduktion der benötigten Rechenzeit. Eine Abschätzung hierfür wird später vorgenommen.
Mit diesen Modifikationen am Verfahren von Sherman-Morrison wurde die Berechnungsvorschrift an den DAP angepaßt.
Im nächsten Kapitel wird zunächst ein Komplexitätsmaß angegeben und das modifizierte Verfahren mit dem ursprünglichen verglichen. In Kapitel 7 werden dann die praktischen Ergebnisse vorgestellt und mit den theoretischen verglichen.

## 6. Die Komplexität des Verfahrens

Um das modifizierte Verfahren nicht nur anhand der gemessenen CPU-Zeiten sondern auch mit einem theoretischen Maß bewerten zu können, muß man zunächst die Komplexität des Verfahrens berechnen. Anschließend kann man es mit dem ursprünglichen Verfahren vergleichen und so entscheiden, ob es tatsächlich eine Verbesserung bringt.
In der Literatur ist es üblich, die Anzahl der Multiplikationen und der Additionen als Komplexitätsmaß heranzuziehen. Dies wird im folgenden ebenfalls zur Berechnung der Komplexität verwendet.

Es gelte für alle weiteren Berechnungen:

$$Dim(A) = n \qquad (A \text{ ist die zu invertierende Matrix})$$

Für das Verfahren von Sherman-Morrison bekommt man dann folgende Abschätzung:

Hilfsvektor:

Anzahl der Multiplikationen: $n^2$

Anzahl der Additionen: $(n - 1)\, n$

k-te Zeile:

Anzahl der Multiplikationen: n

restliche Zeilen:

Anzahl der Multiplikationen: (n - 1) n

Anzahl der Additionen: (n - 1) n

Daraus ergibt sich für jeden Rechenschritt:

Anzahl der Multiplikationen: $2\ n^2$

Anzahl der Additionen: $2\ n^2 - 2\ n$

Da das Verfahren aus n Rechenschritten besteht, erhält man die Komplexität des Verfahrens durch Multiplikation mit n.

Anzahl der Multiplikationen: $2\ n^3$

Anzahl der Additionen: $2\ n^3 - 2\ n^2$

Modifiziert man das Verfahren und startet bei der Berechnung der Inversen mit $A = A^{-1} = I$, so bekommt man bessere Werte. Diese entsprechen den Werten, die man auch bei der Gauß-Elimination erhält.
Es ergeben sich folgende gemittelte Werte:

Hilfsvektor:

Anzahl der Multiplikationen: $n \frac{n}{2}$

Anzahl der Additionen: $(n - 1) \frac{n}{2}$

k-te Zeile:

Anzahl der Multiplikationen: $\frac{n}{2}$

restliche Zeilen:

Anzahl der Multiplikationen: $(n - 1) \frac{n}{2}$

Anzahl der Additionen: $\frac{(n - 1)^2}{2}$

Daraus ergibt sich für jeden Rechenschritt:

Anzahl der Multiplikationen: $n^2$

Anzahl der Additionen: $n^2 - \frac{3}{2} n + \frac{1}{2}$

Für das gesamte Verfahren erhält man:

Anzahl der Multiplikationen: $n^3$

Anzahl der Additionen: $n^3 - \frac{3}{2} n^2 + \frac{1}{2} n$

Dieses Invertierungsverfahren wurde jetzt auf die Struktur des DAP abgebildet. Die Abbildung ist dann optimal, wenn man die hohe Parallelität voll ausnutzen kann. Dazu betrachtet man die Komplexität des Verfahrens in der Realisierung für den DAP: Ohne Einschränkung der Allgemeinheit kann man davon ausgehen, daß gilt: $n = 64 * m$

Dann erhält man für die Realisierung ohne Berücksichtigung der Pivotsuche:

Hilfsvektor:

Anzahl der Multiplikationen: $m \frac{m}{2}$

Anzahl der Additionen: $m ( \frac{m}{2} + 6 \frac{m}{2} )$

( Der Faktor 6 stammt von der Funktion SUMC = Aufsummieren eines Vektors mit $64 = 2^6$ Elementen )

k-te Zeile:

Anzahl der Multiplikationen: $\frac{m}{2}$

Spalten:

Anzahl der Multiplikationen: $\frac{m^2}{2}$

Anzahl der Additionen: $\frac{m^2}{2}$

Daraus ergibt sich für jeden Rechenschritt:

Anzahl der Multiplikationen: $m^2 + \frac{m}{2}$

Anzahl der Additionen: $4m^2$

Realisiert man das Verfahren am DAP mit Pivotsuche über die gesamte Matrix und nimmt man an, daß die Pivotelemente gleichwahrscheinlich über die gesamte Matrix verteilt sind, dann ergibt sich folgende Komplexität:

Hilfsvektor:

Anzahl der Multiplikationen:

$$\frac{\frac{m(m+1)}{2} + (n-m)m}{n} m - \left(\frac{127}{128} m + \frac{1}{128}\right) m = r\,m$$

Anzahl der Additionen ( analog ):

$$\left(\frac{127}{128} m + \frac{1}{128}\right)(1+6)\,m = 7m\left(\frac{127}{128} m + \frac{1}{128}\right) = 7\,rm$$

k-te Zeile:

Anzahl der Multiplikationen: r

Spalten:

Anzahl der Multiplikationen: r m

Anzahl der Additionen: r m

Daraus ergibt sich pro Rechenschritt:

Anzahl der Multiplikationen: r (2m+1)

Anzahl der Additionen: 8 r m

Vergleicht man diese Ergebnisse mit denen ohne Pivotsuche, so erkennt man eine wesentliche Steigerung der Komplexität.
Um diese Steigerung wieder rückgängig zu machen, wurde die Teilpivotsuche eingeführt.

Hierfür errechnet sich folgende Komplexität:

Hilfsvektor:

Anzahl der Multiplikationen: $\left( \frac{m}{2} + 1 \right) m$

Anzahl der Additionen: $7\ m \left( \frac{m}{2} + 1 \right)$

k-te Zeile:

Anzahl der Multiplikationen: $\frac{m}{2} + 1$

Spalten:

Anzahl der Multiplikationen: $m \left( \frac{m}{2} + 1 \right) = \frac{m^2}{2} + m$

Anzahl der Additionen: $\frac{m^2}{2} + m$

Dies ergibt pro Rechenschritt:

Anzahl der Multiplikationen: $m^2 + \frac{3}{2} m + 1$

Anzahl der Additionen: $4\ m^2 + 8\ m$

Für das gesamte Verfahren erhält man:

$$\text{Anzahl der Multiplikationen: } n * \left( m^2 + \frac{3}{2}m + 1 \right)$$

$$\text{Anzahl der Additionen: } n * ( 4\,m^2 + 8\,m )$$

Vergleicht man nun die Komplexität des Verfahrens mit dem der Implementierung, so erhält man folgende Werte:

| | Verfahren von Sherman-Morrison mit Modifikation 1 | Implementierung mit Modifikationen 1,2,3 |
|---|---|---|
| Anzahl der Multiplik. | $n^3$ | $n * \left( m^2 + \frac{3}{2}m + 1 \right)$ |
| Anzahl der Additionen | $n^3 - \frac{3}{2}n^2 + \frac{1}{2}n$ | $n * ( 4\,m^2 + 8\,m )$ |

Mit n = 64*m ergibt sich, daß sowohl die Anzahl der Multiplikationen bei der Implementierung kleiner ist als beim ursprünglichen Verfahren. Das trifft auch auf die Anzahl der Additionen zu.

Dieses Komplexitätsmaß kann man außerdem für den Vergleich der gemessenen Laufzeiten am DAP mit den theoretischen Werten heranziehen.
Der Vergleich wird im nächsten Kapitel durchgeführt.

## 7. Ergebnisse

In diesem Kapitel sollen die Ergebnisse vorgestellt werden, die durch Messungen am DAP ermittelt wurden und sie sollen verglichen werden mit Ergebnissen, die an der Cyber 845 gemessen wurden sowie mit Ergebnissen, die die EGPA-Gruppe mit dem gleichen, an die EGPA-Pyramide angepaßten Verfahren ermittelt hat.

### 7.1 Testmatrizen

Zum Test wurden PEI-Matrizen verwendet [WEST68]. Für die Elemente dieser Matrizen P gilt:

$$p_{ij} = \begin{cases} d & \text{für} \quad i=j \\ 1 & \text{sonst} \end{cases}$$

Daraus ergibt sich für die Elemente von $P^{-1}$ :

$$p^{-1}_{ij} = \begin{cases} \dfrac{d+n-2}{d\,(d+n-2)-(n-1)} & \text{für} \quad i=j \\[2ex] \dfrac{-1}{d\,(d+n-2)-(n-1)} & \text{sonst} \end{cases}$$

Man sieht sofort, daß diese Matrizen singulär werden für

$d = 1$ und $d = 1 - n$.

Damit kann man die Genauigkeit des Verfahrens testen, wenn man in die Nähe dieser Singularitäten kommt.

## 7.2 Implementierung

Für Testzwecke wurde am DAP das Invertierungsverfahren ohne asynchrone Ein-/Ausgabe implementiert und mit den Matrizen aus 7.1 getestet. Das Kernstück des DAP-Programms, der eigentliche Algorithmus, ist verhältnismäßig kurz.

```
      SUBROUTINE SHMTEILPIV(SMAT,EMAT,V,N,NR,NR2)
C
C
C     SMAT=ZU INVERTIERENDE MATRIX
C     EMAT=INVERSE
C     V ENTSPRICHT DEM VEKTOR PIV
C     N IST DIE DIMENSION DER MATRIX
C     NR ENTSPRICHT M
C     NR2 ENTSPRICHT M**2
C
C
      DIMENSION IPIV(,)
      DIMENSION SMAT(,,NR2),EMAT(,,NR2),V(,,NR),VH(,)
      LOGICAL PIV(,),IPL(,)
      PIV=.TRUE.
C
C
C     ALLE SPALTEN NOCH NICHT VERWENDET
C
C
      DO 2 K=1,N
      VH=0.
      IF(K.NE.1) GOTO 3
C
C
C     FUER K=1 HILFSVEKTOR = 1. SPALTE DER MATRIX
C
C
      DO 21 KK=1,NR
      VH(,KK)=SMAT(,1,KK)
21    CONTINUE
      GOTO 4
C
C
C     BERECHNUNG DES HILFSVEKTORS FUER K>1
C
C
```

```
3       IVZ=0
        K1=K
6       IVZ=IVZ+1
        K1=K1-64
        IF(K1) 5,5,6
5       K1=K1+64
        DO 70 KK=1,NR
        V(,,KK)=0.
        IF(.NOT.ANY(PIV(,KK))) GOTO 72
        VH(PIV(,KK),KK)=MATC(SMAT(,K1,(IVZ-1)*NR+KK))
72      IF(ALL(PIV(,KK))) GOTO 70
        V(.NOT.MATR(PIV(,KK)),KK)=MATR(SMAT(,K1,(IVZ-1)*NR+KK))
70      CONTINUE
        DO 71 KK=1,NR
        DO 71 KK1=1,NR
        IF(ALL(PIV(,KK1))) GOTO 71
        VH(,KK)=VH(,KK)+SUMC(EMAT(,,(KK1-1)*NR+KK)*V(,,KK1))
71      CONTINUE
4       K1=K+64
        IF(K1.GT.4096) K1=4096
        IPL=.FALSE.
        VIPL=0.
C
C
C       PIVOTBILDUNG
C
C
74      IPL=FRST(MAXP(ABS(VH),PIV.AND.ELSL(1,K1)))
        VIPL=VH(IPL)
        IF(VIPL.NE.0.) GOTO 73
        K1=4096
        GOTO 74
73      PIV(IPL)=.FALSE.
        IIPL=ELN(IPL)
        IIPL1=IIPL/64
        IF(64*IIPL1.NE.IIPL) IIPL1=IIPL1+1
        IIPL2=IIPL-64*(IIPL1-1)
        DO 22 KK1=1,NR
C
C
C       BERECHNUNG K-TE ZEILE
C
C
        IF(ALL(PIV(,KK1))) GOTO 24
        EMAT(ROW(IIPL2).AND.MATR(.NOT.PIV(,KK1)),(KK1-1)*NR+IIPL1)=
     1EMAT(,,(KK1-1)*NR+IIPL1)/VIPL
        EMAT(.NOT.ROW(IIPL2).AND.MATR(.NOT.PIV(,KK1)),(KK1-1)*NR
     1+IIPL1)=EMAT(,,(KK1-1)*NR+IIPL1)-MATC(VH(,IIPL1))*MATR(EMAT
     2(IPL(,IIPL1),,(KK1-1)*NR+IIPL1))
24      DO 23 KK10=1,NR
C
C
C       BERECHNUNG DER SPALTEN
C
C
```

```
      IF(KK10.EQ.IIPL1) GOTO 23
      IF(ALL(PIV(,KK1))) GOTO 23
      EMAT(MATR(.NOT.PIV(,KK1)),
     1(KK1-1)*NR+KK10)-EMAT(,,(KK1 1)*NR+KK10)-MATC(VH(,KK10))*
     2MATR(EMAT(IPL(,IIPL1),,(KK1 1)*NR+IIPL1))
23    CONTINUE
22    CONTINUE
      IPIV(IPL)=K
2     CONTINUE
      DO 50 K=1,N
      KVZ=K/64
      IF(64*KVZ.NE.K) KVZ=KVZ+1
      IVZ=IPIV(K)/64
      IF(64*IVZ.NE.IPIV(K)) IVZ=IVZ+1
C
C
C     ZEILENPERMUTATION BEHEBEN
C
C
      DO 50 KK=1,NR
      SMAT(IPIV(K)-64*(IVZ-1),,(KK-1)*NR+IVZ)=
     1EMAT(K-64*(KVZ-1),,(KK-1)*NR+KVZ)
50    CONTINUE
      RETURN
      END
```

Im Rahmen einer Studienarbeit wird zur Zeit um den Kern herum eine Umgebung erstellt, die dem Benutzer eine einfache Handhabung des Invertierungsverfahrens einschließlich asynchroner Ein-/ Ausgabe ermöglicht.

## 7.3 Meßergebnisse am DAP ohne asynchrone Ein-/Ausgabe

Bei Vollausnutzung der DAP - Prozessor - Matrix, d.h.
dim (P) = 64*m, ergaben sich folgende Laufzeiten:

| Größe der Matrix | Laufzeit in Sekunden | |
|---|---|---|
| | bei voller | teilweiser Pivotsuche |
| 64 x 64 | 0.245 | - |
| 128 x 128 | 1.312 | 1.049 |
| 192 x 192 | 3.859 | 2.746 |
| 256 x 256 | 8.541 | 5.673 |
| 320 x 320 | 16.014 | 10.167 |
| 384 x 384 | 26.936 | 16.563 |
| 448 x 448 | 41.960 | 25.199 |
| 512 x 512 | 61.744 | 36.411 |
| 768 x 768 | 196.824 | 113.771 |

Für nicht voll ausgelastete DAP-Prozessor-Matrizen ergeben sich folgende Werte:

| | Laufzeit in Sekunden | |
|---|---|---|
| | bei voller | teilweiser Pivotsuche |
| 100 x 100 | 1.027 | 0.766 |
| 150 x 150 | 3.017 | 1.909 |
| 200 x 200 | 6.674 | 3.814 |
| 250 x 250 | 8.345 | 5.478 |
| 300 x 300 | 15.019 | 9.174 |
| 350 x 350 | 24.555 | 14.186 |
| 400 x 400 | 37.465 | 20.708 |
| 450 x 450 | 54.263 | 28.929 |
| 500 x 500 | 60.310 | 34.977 |

Ohne Modifikation 1 wäre der Rechenaufwand noch wesentlich höher. Er wäre etwa doppelt so groß wie in der 1.Spalte der Tabelle angegeben (vgl. S.57).
Wie man durch den Vergleich der beiden Spalten in obiger Tabelle sieht, spart man durch die 3. Modifikation (Teilpivotisierung) bei großen Matrizen etwa 50% Rechenzeit. Diese Ergebnisse wurden ohne asynchrone Ein-/Ausgabe erzielt, da die Matrizen noch komplett in den DAP-Speicher paßten.
Mit dim (P) = 768 war der Speicher des DAP durch dieses Berechnungsverfahren voll ausgenutzt. Für größere Matrizen muß ein asynchrones Nachladen der Daten vorgesehen werden.

## 7.4 Vergleich mit Ergebnissen an anderen Rechnern

Da das Verfahren von Sherman - Morrison auch für das EGPA - System angepaßt und implementiert wurde, lassen sich die Ergebnisse direkt vergleichen.
Der Speicherausbau bei EGPA läßt nur die Berechnung der Inversen bei verhältnismäßig kleinen Matrizen zu.
Für eine 100 x 100 Matrix wurde eine Zeit zwischen 13 und 14 Sekunden gemessen, für eine 200 x 200 Matrix 103 - 104 Sekunden.
Damit ergibt sich ein Verhältnis der CPU-Zeiten von DAP zu EGPA bei der kleinen Matrix von 1 : 17, bei der großen von 1 : 27.
Das Verfahren wurde auch an der Cyber 845 im Rahmen einer Studienarbeit implementiert.
Die Ergebnisse und den Vergleich mit dem DAP kann man den folgenden Tabellen entnehmen (alle Angaben in Sekunden).

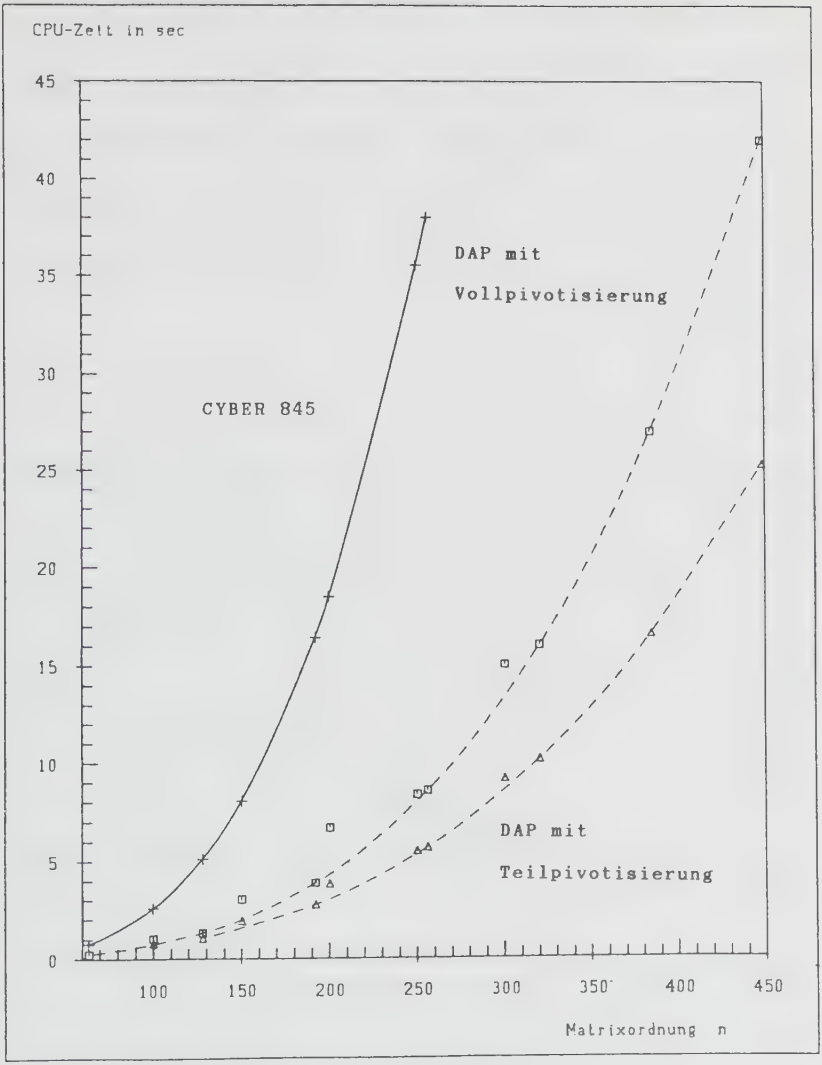
| n / Version | 50 | 64 | 100 | 128 | 150 | 200 | 256 |
|---|---|---|---|---|---|---|---|
| DAP mit Vollpivot | - | 0.245 | 1.027 | 1.312 | 3.017 | 6.674 | 8.541 |
| DAP mit Teilpivot | - | - | 0.776 | 1.049 | 1.909 | 3.814 | 5.673 |
| Cyber845 | 0.424 | 0.780 | 2.563 | 5.089 | 8.029 | 18.495 | 37.969 |
| EGPA | < 2 | | 13 - 14 | | | 103-104 | |

Diese Werte ergeben folgenden speed-up:

| Verhältnis \ n | 50 | 64 | 100 | 128 | 150 | 200 | 256 |
|---|---|---|---|---|---|---|---|
| Cyber/DAP-Vollpivot | - | 3.18 | 2.5 | 3.88 | 2.66 | 2.77 | 4.45 |
| Cyber/DAP-Teilpivot | - | - | 3.35 | 4.85 | 4.21 | 4.85 | 6.69 |
| Cyber/EGPA | > 0.21 | | 0.19 | | | 0.18 | |

Aus der Tabelle ist deutlich zu ersehen, daß der DAP mit zunehmender Matrixgröße der Cyber 845 immer überlegener wird und daß dieser speed-up nur bei voll ausgelastetem Prozessorfeld am DAP erzielt wird.

Graphisch dargestellt ergeben sich damit folgende Rechenzeitkurven:


CPU-Zeit in sec
45
40
35
30
25
20
15
10
5
0
100
150
200
250
300
350
400
450
Matrixordnung n
CYBER 845
DAP mit
Vollpivotisierung
DAP mit
Teilpivotisierung

Vergleicht man das Verfahren von Sherman-Morrison mit anderen Invertierungsverfahren aus der NAG-Bibliothek an der Cyber, so sieht man das es etwa doppelt so schnell ist wie das Verfahren von Crout und etwas langsamer als das Zerlegungsverfahren von Cholesky [SCHW85]. Da sich das Cholesky-Verfahren nur für symetrische positiv definite Matrizen eignet, ist das Verfahren von Sherman-Morrison auch an der Cyber 845 das schnellste Invertierungsverfahren für beliebige nichtsinguläre Matrizen.

## 7.5 Fehlerabschätzung

Messungen zur Fehlergenauigkeit haben ergeben, daß der maximale Fehler bei gut konditionierten Matrizen bei $10^{-6}$ liegt.
Sind die Matrizen schlecht kondinioniert (nahe an der Singularität), dann nimmt der Fehler sehr stark zu und steigt auf $10^{-3}$. Diese Ergebnisse wurden erzielt durch entsprechendes Ändern von d in den obigen Testmatrizen.
Die Größe des Fehlers war dabei nur unwesentlich abhängig von der Dimension der Matrix. Diese Aussage läßt sich nicht auf beliebige Matrizen verallgemeinern.
Man kann jedoch eine Fehlerabschätzung wie beim Gauß-Verfahren machen.

Da das Verfahren von Sherman-Morrison äquivalent zur Gauß-Elimination ist [WEST68], läßt sich eine obere Fehlerschranke für das Verfahren von Sherman-Morrison angeben [BUNS85]:

$$f \leq n \; a \; g \; eps$$

Dabei ist

- $n$ Dimension der Matrix A
- $a$ $\max |a_{ij}|$
- $g$ $\max |a^{(k)}_{ij}|$
- $eps$ Maschinengenauigkeit

## 7.6 Vergleich der theoretischen und praktischen Ergebnisse

Da die typischen Rechenzeiten für Matrizenoperationen am DAP als absolute Werte vorliegen, läßt sich hieraus theoretisch die Rechenzeit am DAP berechnen und mit den tatsächlich gemessenen Ergebnissen vergleichen.
Da n Rechenschritte für die Invertierung benötigt werden, ergibt sich aus den Komplexitätsbetrachtungen die Anzahl der notwendigen Matrizenoperationen bei der Teilpivotsuche.

Anzahl der Multiplikationen:

$$n \left( m^2 + \frac{3}{2}m + 1 \right) = 64 \left( m^3 + \frac{3}{2}m^2 + m \right)$$

Anzahl der Additionen:

$$n \left( 4m^2 + 8m \right) = 64 \left( 4m^3 + 8m^2 \right)$$

Eine Matrizenmultiplikation (elementweise) dauert am DAP $273.5 * 10^{-6}$ Sekunden, ein Matrizenaddition $157.3 * 10^{-6}$ Sekunden. Das ergibt folgende Tabelle:

| n | m | Rechenzeit ( Sek.) | | gem. Rechenzeit / theor. Rechenzeit |
|---|---|---|---|---|
| | | theoretisch | gemessen | |
| 256 | 4 | 5.47 | 5.67 | 1.04 |
| 320 | 5 | 9.97 | 10.17 | 1.02 |
| 384 | 6 | 16.42 | 16.56 | 1.01 |
| 448 | 7 | 25.17 | 25.20 | 1.00 |
| 512 | 8 | 36.55 | 36.41 | 0.99 |
| 768 | 12 | 115.77 | 113.77 | 0.99 |

Die theoretischen Werte sind fast identisch mit den gemessenen. Die im Programm zusätzlich auftretenden logischen Operationen sind vernachlässigbar. Bei größerem n tritt sogar der Fall ein, daß der theoretische Wert größer als der gemessene wird. Das liegt daran, daß die Zeiten für die Matrizenmultiplikation bzw. Addition vom DAP-Prototyp stammen.
Für die erste Kundenmaschine wurden Modifikationen an der Maschinensprache vorgenommen. So wurde insbesondere ein "add to store"-Befehl eingeführt, der in 1.5 Zyklen ausgeführt wird gegenüber 2 Zyklen für den Befehl "add and store".

## 7.7 Abschätzung der benötigten Rechenzeit

Die bisherigen Ergebnisse kann man dazu heranziehen, um dem Benutzer dieses Invertierungsverfahrens eine einfache Formel an die Hand zu geben, mit der er seine benötigte Rechenzeit am DAP abschätzen kann.
Die Anzahl der Gesamtoperationen ( Multiplikation und Addition ) ist 64 ( $5m^3 + 9.5m^2 + m$ ). Nimmt man als mittlere Befehlsausführungszeit $200 * 10^{-6}$ Sekunden an, so bekommt man eine obere Schranke für die Rechenzeit. Dies kann man auch der folgenden Tabelle entnehmen:

| n | m | Rechenzeit (sek.) | obere Schranke (sek.) |
|---|---|---|---|
| 256 | 4 | 5.67 | 6.09 |
| 320 | 5 | 10.17 | 11.10 |
| 384 | 6 | 16.56 | 18.28 |
| 448 | 7 | 25.20 | 28.00 |
| 512 | 8 | 36.41 | 40.65 |
| 768 | 12 | 113.77 | 146.68 |

Damit kann der Benutzer seine voraussichtliche Rechenzeit mit der Formel

$$t_{DAP} = 128 * 10^{-4} ( 5m^3 + 9.5m^2 + m ) \quad [sec]$$

abschätzen.

# 8. Asynchrone Ein-/Ausgabe und Sherman-Morrison-Verfahren

Nachdem in den vorhergehenden Kapiteln die asynchrone Ein-/Ausgabe und das Verfahren zur Invertierung von Matrizen einzeln analysiert, modifiziert und bewertet wurden, soll in diesem Kapitel die Synthese dieser beiden Methoden dargestellt und bewertet werden.

Um das Zusammenspiel der beiden Komponenten zu verstehen, muß man zunächst das Scheduling im DAP, das zu einem Multistreaming führt, betrachten.

Im DAP gibt es 3 Warteschlangen, die Aufträge mit verschieden großem Hauptspeicherbedarf enthalten. Es gibt je eine Warteschlange für Aufträge mit 8 Mbyte, 6 Mbyte und 2 Mbyte Speicherbedarf bei einer Gesamtspeichergröße von 8 Mbyte.

Die Aufträge aus der 8 Mbyte Warteschlange werden nur zu bestimmten Zeiten (z.B. nachts) abgearbeitet. Diese Betriebsweise im Single-User-Betrieb soll uns im weiteren nicht interessieren.

Im Normalbetrieb und bei Aufträgen sowohl in der 2- als auch in der 6-Mbyte-Warteschlange teilen sich je ein Auftrag aus jeder Warteschlange den Speicher des DAP. Diese Aufträge werden im Multistreaming abgearbeitet.

Dies bedeutet, daß immer dann der zweite Auftrag vom Zustand Warten in den Zustand Rechnend übergeht, wenn der erste Auftrag die Regie abgibt, um z.B. zurück in den Host-Teil des Programms zu verzweigen. Dieser zweite Auftrag bleibt so lange rechnend bis er von sich aus die Regie wieder abgibt. Es erfolgt also keine Unterbrechung der Aufträge durch das Betriebssystem.

Diesen Vorgang der wechselseitigen Abhängigkeit zweier Aufträge kann man für die Kombination des Invertierungsverfahrens mit der asynchronen Ein-/Ausgabe ausnutzen.
Betrachten wir folgendes Ablaufdiagramm:

Dabei ist x die Zeit, in der der Host oder der DAP vom jeweiligen Auftrag belegt ist, a die Zeit, in der asynchrone Ein-/Ausgabe während des eigenen Auftrags und b die Zeit, in der asynchrone Ein-/Ausgabe während des Fremdauftrags abläuft.
In Kapitel 3 wurde festgestellt, daß der asynchrone Transfer einer Matrix ca. 200 ms dauert. Das Berechnungsverfahren arbeitet mit Blöcken von jeweils 64 Spalten. Diese Spalten sind in m Matrizen durch das Cracking abgespeichert.
Dabei ist m = dim (A)/64. Der asynchrone Transfer dauert für diese 64 Spalten 200*m Millisekunden.

Steuert man seinen Auftrag durch entsprechende Angaben für den Scheduler so, daß er jeweils einen Partnerauftrag mit hohem Rechenzeitbedarf bekommt, so wird die asynchrone Ein-/Ausgabe für den eigenen Auftrag zum größten Teil dann abgewickelt, wenn der Partnerauftrag den DAP belegt.
Kommt anschließend der eigene Auftrag an die Regie, stehen die neuen Daten entweder ganz oder zum größten Teil zur Verfügung. Die Bearbeitung kann sofort beginnen.
Der Vorteil des Benutzers bei dieser Synthese von asynchroner Ein-/Ausgabe und Invertierungsverfahren liegt in der verkürzten Verweilzeit des Auftrags im Rechner. Die benötigte CPU-Zeit ändert sich nicht.
Stellt man den Ablauf von synchronem und asynchronem Verfahren gegenüber, dann kann man berechnen, wie groß die Ersparnis in der Verweilzeit ist.

Synchroner Fall:

1. Berechnung von 768 Spalten.

2. Verzweigen in den Host und Auslagern von 768 Spalten und Nachladen der nächsten 768 Spalten, falls noch vorhanden. Sprung nach 1.

Asynchroner Fall:

1. Berechnung von 786 Spalten und nach jeweils 64 Spalten Anstoß zum Auslagern von Blöcken mit 64 Spalten

2. Verzweigung in den Host und Nachladen weiterer 768 Spalten, falls noch vorhanden. Sprung nach 1.

Der Unterschied asynchron/synchron liegt darin, daß bereits während der Berechnung das Auslagern nicht mehr benötigter Spalten initiiert wird. Der Host muß dann nur noch die neuen Daten nachladen.
Die Zeitersparnis läßt sich damit wie folgt berechnen:

Nimmt man als Berechnungszeit die in Kapitel 7 angegebene obere Schranke

$$t_{DAP} = 128*10^{-4}\ (5m^3 + 9.5m^2 + m)\ \text{Sekunden.}$$

dann erhält man pro Rechenschritt eine mittlere Zeit

$$t_{Schritt} = \frac{t_{DAP}}{64m} = 2*10^{-4}\ (5m^2 + 9.5m + 1)\ \text{Sekunden.}$$

Für einen Block mit 768 Spalten benötigt man dann, da im Mittel n/2 Spalten berechnet werden

$$t_{768} \quad \frac{t_{Schritt}}{\frac{n}{2*768}} = 48*10^{-4}\ (5m + 9.5 + \frac{1}{m})\ \text{Sekunden.}$$

Für große m gilt dann:

$$t_{768} = 2.4*10^{-2}*m \text{ Sekunden.}$$

Da bereits nach der Berechnung von 64 Spalten mit dem Auslagern begonnen werden kann, spart man durch dieses Verfahren pro Block mit 768 Spalten

$$\frac{11}{12} t_{768} = 2.2*10^{-2}*m \text{ Sekunden.}$$

Der asynchrone Transfer von 64 Spalten dauert aber bereits $20*10^{-2}*m$ Sekunden. Das bedeutet das während 11/12-tel der Rechenzeit asynchroner E/A-Verkehr möglich ist und vollkommen als ersparte Verweilzeit angesehen werden kann.

Ergebnis:

Für das gesamte Verfahren ergibt sich eine verkürzte Verweilzeit von

$$t_{SPAR} = \frac{11}{12} t_{DAP}$$

Dies bedeutet, daß 11/12-tel der Rechenzeit auch dazu verwendet werden können, Datentransfer gleichzeitig auszufuehren. Damit wird die Verweilzeit des Auftrags im Rechner in etwa um die Rechenzeit verkuerzt.

## 9. Zusammenfassung und Ausblick

In dieser Arbeit wurden am Beispiel der Invertierung von Matrizen die Probleme analysiert, die auftreten, wenn die Matrizen groß sind und nicht komplett im Speicher des DAP untergebracht werden können.
Alle bisherigen Problemlösungen liefen auf ein Nacheinander von Bearbeitung eines Teils der Daten, Auslagern und Nachladen von Daten, Bearbeitung eines Teils der Daten usw. hinaus.
Das hier vorgeschlagene Verfahren hat den Vorteil, daß

1. Nur ein klar adressierbarer Teil der Daten pro Rechenschritt gebraucht wird und
2. Nicht benötigte Daten bereits während des Rechenvorgangs ausgelagert werden können.

Dieses Prinzip läßt sich nicht nur auf die Invertierung von Matrizen anwenden, sondern auch auf andere Operationen auf großen Matrizen. Ob dafür bereits geeignete Algorithmen vorhanden sind oder neu entwickelt werden müssen, kann in dieser Form nicht generell gesagt werden.
Das Prinzip der asynchronen Ein-/Ausgabe ist sicherlich nicht beschränkt auf den ICL-DAP, es läßt sich unter Umständen auch auf andere unkonventionelle Rechner, die an einen Host angeschlossen sind, übertragen.

Literaturverzeichnis


[BUNS85] W.Bunse/A.Bunse-Gerstner
Numerische lineare Algebra
Teubner, Stuttgart, 1985

[CHEN72] T.C.Chen
Automatic Computation of Exponentials, Logarithms,
Ratios and Square Roots
IBM J.Res.Develop. 16, 1972, S. 380

[FEIL] M. Feilmeier
Parallel Numerical Algorithms
Interner Bericht, Institut für Rechentechnik
TU Braunschweig

[FORT79] DAP: FORTRAN Language
ICL, 1979

[FORT82] DAP FORTRAN Block Transfer System
ICL, 1982

[GOST79] R.W. Gostick
DAP Algorithms for standard numerical functions
ICL Technical Journal 5/79

[HENN] W. Henning
Matrixinversion durch ein Spaltenersetzungsverfahren
auf dem Multiprozessorsystem EGPA
Interner Vortrag, Institut für math. Maschinen und
Datenverarbeitung III, Universität Erlangen, Erlangen

[HOJE81] R.W. Hockney, C.R. Jesshope
Parallel Computers
A. Hilger Ltd., Bristol, 1981

[HOSS81] F. Hossfeld
Parallele Algorithmen
Vorlesungsskriptum, Jülich, 1981

[MEGG62] J.E. Meggitt
Pseudo Division and Pseudo Multiplication Processes
IBM J.Res.Develop. 6, 1962, S.210

[PARK77] D. Parkinson
An Introduction to Array Processors
Systems International 11/77

[REIN83] K.D. Reinartz
Bitalgorithmen zur schnellen Berechnung transzendenter Funktionen auf Parallelrechnern
Parallel Computing 83, North Holland, 1983

[SCHW85] M. Schwegler
Implementierung und Bewertung eines Invertierungsverfahrens für Matrizen an der CYBER 845 und Vergleich mit Ergebnissen an DAP und EGPA
Studienarbeit, Erlangen, 1985

[SHMO49] J. Sherman, W.J. Morrison
Adjustment of an Inverse Matrix Corresponding to Changes in the Elements of a Given Column or a Given Row of the Original Matrix
Ann. Math. Stat. 20, 1949, S. 621

[SHMO50] J. Sherman, W.J. Morrison
Adjustment of an Inverse Matrix Corresponding to a Change in one Element of a Given Matrix
Ann. Math. Stat. 21, 1950, S. 124

[STON75] Stone
Introduction to Computer Architecture
SRA, 1975

[WEBB80] S.J. Webb
Solution of elliptic partial differential equations
on the ICL DAP
ICL Technical Journal, 2, 1980

[WEST68] Joan R. Westlake
A Handbook of Numerical Matrix Inversion and Solution
of Linear Equations
Wiley & Sons, New York, 1968

## Lebenslauf

Werner Erhard, Praterstr. 9, 85 Nürnberg 80
geb. 24.8.1949 in Nürnberg,
verheiratet seit August 1973, keine Kinder.

| | |
|---|---|
| 1955 - 1959 | Volksschule in Nürnberg |
| 1959 - 1968 | Wirtschaftsgymnasium in Nürnberg |
| 1968 - 1974 | Mathematik-Studium an der Universität Erlangen mit Abschluß Diplom-Mathematiker |
| Seit 1.9.1974 | Wiss. Angestellter am Institut für Mathematische Maschinen und Datenverarbeitung<br>Bis 1980 Mitarbeit in der Informatik-Rechner-abteilung als Systembetreuer TR440<br>Ab 1981 Mitarbeit am Lehrstuhl für Rechner-architektur und -Verkehrstheorie<br>( IMMD VII, Prof Dr. U. Herzog ) |

In letzter Zeit sind folgende Arbeitsberichte des Instituts für Mathematische Maschinen und Datenverarbeitung erschienen:

Band 13

Nr. 1 Schrammel, D., Bayer, W., Gliemann, H.-P., Heigl, M., Mertens, P.:
Ein Leistungs- und Wirtschaftlichkeitsvergleich zwischen Klein- und Großcomputern
(Januar 1980)

Nr. 2 Göttler, H.:
Semantische Modelle
(März 1980)

Nr. 3 Fromm, H.J.:
Zur Modellierung der Speicherinterferenz bei hierarchisch organisierten Multiprozessorsystemen
(April 1980)

Nr. 4 Jäpel, D.:
Klassifikatorbezogene Merkmalsauswahl
(Juli 1980)

Nr. 5 Bodendorf, F., Mertens, P., Haas, R., Weber, G., Wolf, G.:
SAMBA - ein Methodenbankrahmen um das Statistik-Paket SPSS
(Juli 1980)

Nr. 6 Bartsch, B.:
Inferenz und Analyse spezieller Graphgrammatiken für die syntaktische Mustererkennung
(September 1980)

Nr. 7 Wurm, F.X.:
Auftragssystem für eine Mehrprozessoranlage
(Oktober 1980)

Nr. 8 Schedelbeck, A.:
Vergleich der beiden höheren Realzeitprogrammiersprachen ADA und PEARL
(Oktober 1980)

Nr. 9 Jahresbericht 1980 der Informatik
(März 1981)

Band 14
=======

Nr. 1 Pattern Recognition
(Research at Lehrstuhl für Informatik 5)
(Februar 1981)

Nr. 2 Bunke, H.:
Analyse elektrischer Schaltpläne
(März 1981)

Nr. 3 Linster, C.-U.:
SYMPOS/UNIX - Ein Betriebssystem für homogene Polyprozessorsysteme
(Juni 1981)

Nr. 4 Akyildiz, J.F., Bolch, G.:
Analytic Solution Techniques for Queueing Network Models of Computer Systems
(Juli 1981)

Nr. 5 Lindstedt, W.G.:
Ein Verfahren zur Erstellung portabler algorithmischer Laufzeitprogramme
(Juli 1981)

Nr. 6 Vollmar, R.:
Zum Begriff des Parallelismus bei Polyautomaten
(Juli 1981)

Nr. 7 Luft, A.L.:
Rationaler Sprachgebrauch und orthosprachliche Standardisierung als Grundlagen zuverlässiger Software-Entwicklung und -Dokumentation
(Januar 1982)

Nr. 8 Händler, W.:
PROCEEDINGS OF A WORKSHOP ON TAXONOMY IN COMPUTER ARCHITECTURE
(Februar 1982)

Nr. 9 Beth, Th., Hess, P., Wirl, K.:
Materialien zur Kryptographie
(März 1982)

Nr. 10 Jahresbericht 1981 der Informatik
(März 1982)

Band 15
=======

Nr. 1 Wittmann, A.:
Ein Mechanismus für die Synchronisation paralleler Prozesse
(Februar 1982)

Nr. 2 Maehle, E.:
Fehlertolerantes Verhalten in Multiprozessoren - Untersuchungen zur Diagnose und Rekonfiguration
(März 1982)

Nr. 3 Bathelt, P.:
Vergleich von Synchronisationsmechanismen
(Mai 1982)

Nr. 4 Keramidis, S.:
Eine Methode zur Spezifikation und korrekten Implementierung von asynchronen Systemen
(Juni 1982)

Nr. 5 Fromm, H.J.:
Multiprozessor-Rechenanlagen:
Programmstrukturen, Maschinenstrukturen und Zuordnungsprobleme
(Mai 1982)

Nr. 6 Bley, H.:
Vorverarbeitung und Segmentierung von Stromlaufplänen unter Verwendung von Bildgraphen
(Juli 1982)

Nr. 7 Hchl, W.:
Informatik-Sammlung
Katalog mit historischen Erläuterungen
(August 1982)

Nr. 8 Messerer, M.:
Ein neuer Ansatz zur Parallelisierung von Compilern
(August 1982)

Nr. 9 Grosch, J.:
Eine Programmiersprache mit mengentheoretischen Konstrukten und deren effiziente Implementierung
(August 1982)

Nr. 10 Kleinöder, W.:
Stochastische Bewertung von Aufgabenstrukturen für hierarchische Mehrrechnersysteme
(August 1982)

Band 15 Fortsetzung
==========

Nr. 11 Kneißl, F.:
Realisierung von Makro-Datenflußmechanismen auf hierarchischen Mehrrechnersystemen
(August 1982)

Nr. 12 Brendel, W., Fan, Z., Klar, R., Schmielau, W.:
ERES 82
Handbuch und Fallstudie
Handbook and Case Study
(Oktober 1982)

Nr. 13 Pelz, K.:
Portable Codegenerierung in Übersetzern für Prozeß-Programmiersprachen
(Oktober 1982)

Nr. 14 Sommer, U., Steinchen, J.:
Ein Kleinrechner-Grafiksystem zur Dokumentation von EDV-Abläufen
(Oktober 1982)

Nr. 15 Hein, H.-W.:
Das Erlanger Spracherkennungssystem
(November 1982)

Nr. 16 Bolch, G., Bruchner, W.:
Analytische Modelle für symmetrische Mehrprozessoranlagen mit dynamischen Prioritäten
(Dezember 1982)

Nr. 17 Jahresbericht 1982 der Informatik
(März 1983)

Band 16
=======

Nr. 1 Gall, R.:
Formale Beschreibung des inkrementellen Programmierens-im-Großen mit Graph-Grammatiken
(März 1983)

Nr. 2 König, R.:
Beiträge zur algebraischen Theorie der formalen Sprachen
(März 1983)

Nr. 3 Welbourn, D.B.:
The Design of Mechanical Components and the Development of DUCT:
17 Years of CAD/CAM in Cambridge University
(April 1983)

Nr. 4 Woitok, R.:
Die System-Implementierungs-Sprache ALICE
(Mai 1983)

Nr. 5 Schneider, S., Schwab, P., Renninger, W.:
Wesen, Vergleich und Stand von Software zur Produktion von Systemen der computergestützten Unternehmensplanung
(Juli 1983)

Nr. 6 Händler, W., Herzog, U., Hofmann, F., Schneider, H.J. und Mitarbeiter:
Projektabschlußbericht 1978 - 1982 Erlangen General Purpose Array
(Juli 1983)

Nr. 7 Mackert, L.:
Modellierung, Spezifikation und korrekte Realisierung von asynchronen Systemen
(Juli 1983)

Nr. 8 Reitenspieß, M.:
Sprachkonstrukte zur Spezifikation und korrekten Implementierung von Schutzproblemen
(Juli 1983)

Nr. 9 Weber, K.:
Modellierung von Fehlverhalten mit Berücksichtigung paralleler Abläufe
(August 1983)

Nr. 10 Dietsch, H.:
Ein verteiltes Kommunikationssystem für einen Verbund aus lokalen Mikrorechner-Netzwerken
(Dezember 1983)

Band 16 Fortsetzung
==========

Nr. 11 Maehle, E., Schmitter, E.:
Workshop Fehlertolerante Mehrprozessor- und Mehrrechnersysteme
(Dezember 1983)

Nr. 12 Volkert, J.:
Multiplikative Abstandsfunktionen und deren Anwendung in der Bildverarbeitung
(Dezember 1983)

Nr. 13 Steinbauer, D.:
Transaktionen als Grundlage zur Strukturierung und Integritätssicherung in Datenbank-Anwendungssystemen
(Dezember 1983)

Nr. 14 Wilke, P.:
Zusammenhänge und Unterschiede zwischen Graph-Grammatiken und Petri-Netzen sowie verwandter Systeme
(Dezember 1983)

Nr. 15 Jahresbericht der Informatik 1983
(März 1984)

Band 17
=======

Nr. 1 Eberlein, W.:

Architektur technischer Datenbanken für integrierte Ingenieursysteme
(Februar 1984)

Nr. 2 Mertens, P., Heigl, M.:

Neuere Entwicklungen der computergestützten Produktionsplanung, Eignung - Verbindungen - Entwicklungspfade
(April 1984)

Nr. 3 Stoyan, H.:

Maschinen- unabhängige Code-Erzeugung als semantikerhaltende beweisbare Programmtransformation
(Juli 1984)

Nr. 4 Brendel, W.:

Funktionaler Entwurf und automatische Synthese hochintegrierter Schaltkreise
(August 1984)

Nr. 5 Brietzmann, A.:

Semantische und pragmatische Analyse im Erlanger Spracherkennungsprojekt
(Oktober 1984)

Nr. 6 Hoppe, B.:

Deterministisches Zuordnen vielfach durchlaufener Aufgabensysteme in Multiprozessorsystemen
(Oktober 1984)

Nr. 7 Fumy, W., Posch, S., Rieß, H.P.:

Materialien zur Codierungstheorie III
(Dezember 1984)

Nr. 8 Akyildiz, J. F.:

Leistungsanalyse von Multiprozessorsystemen mit Prozeßkommunikation
(Dezember 1984)

Nr. 9 Jahresbericht der Informatik 1984

(März 1985)

Band 18
=======

Nr. 1 Siegmann, H.:

Wesen, Vergleich und Stand von zehn ausgewählten kleinrechnerorientierten Planungssprachen
(Februar 1985)

Nr. 2 Rathke, M.:

Parallelisieren ordnungserhaltender Programmsysteme für hierarchische Multiprozessorsysteme
(Juni 1985)

Nr. 3 Geus, L.:

Parallelisierung eines Mehrgitterverfahrens für die Navier-Stokes-Gleichungen auf EGPA-Systemen
(Juli 1985)

Nr. 4 Hellmold, K.-U.:

Der Simulationsrechner SIMPLEX
Eine Multiprozessorkonfiguration zur Simulation von zeitdiskreten Systemen auf GPSS-FORTRAN-Basis
(September 1985)

Nr. 5 Klebes, G.:

Entwicklung eines botschaftsorientierten Synchronisationsverfahrens für parallele Prozesse und seine Realisierung im Rahmen eines Realzeitprogrammiersystems
und
Rupprecht, G.:

Ein Konzept zur Integration von allgemeinen Sprachkonstrukten zur Synchronisation und von gemeinsamen abstrakten Datentypen mehrerer Prozesse in Modula-2 und seine Implementierung

Nr. 6 Schön, D.:

Technisches Informationssystem und Entwurf integrierter Ingenieurdatenbanken
(Oktober 1985)

Nr. 7 Vasconcelos, A.J.V. de:

Berechnung nichtstationärer Vorgänge in Drehstromnetzen mit Raumzeigerkomponenten auf einem Multiprozessor
(Dezember 1985)

Nr. 8 Baader, F.:

Die S-Varietät DS und einige Untervarietäten
(Dezember 1985)

Nr. 9 Jahresbericht der Informatik 1985
(März 1986)

Offsetdruck · Fotodruck · Hogl · Hauptstraße 109 · 8520 Erlangen · Tel.09131/22990